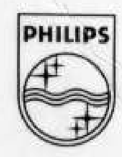
# PHILIPS




# MSX2-BASIC + MSX-DOS




CE MANUEL CONTIENT TOUTES LES INSTRUCTIONS DU MSX2-BASIC ET DU MSX-DOS

**A. SICKLER**
**A. VAN UTTEREN**

New Media Systems

Albert Sickler
Aaldrik van Utteren

# MSX2-BASIC + MSX-DOS

Edition Philips



Kluwer Technische Boeken b.v.
Deventer - Anvers

Traduction: Expertrans, Zoetermeer, Holland



Première édition 1986

# INTRODUCTION AU BASIC

## 1 LES PREMIERS PAS: LA COMMANDE PRINT ET LES CALCULS

**PRINT pour l'exécution d'opérations directes**

Quand nous mettons en marche l'ordinateur, celui-ci est immédiatement prêt pour travailler sous MSX-BASIC.
Pour montrer cela, nous entrons:

```
PRINT 2+3
```

L'expression PRINT 2+3 apparaît maintenant sur l'écran. L'ordinateur réagira seulement à cette instruction quand nous appuierons sur la touche RETURN.

PRINT veut dire 'imprimez' ou plutôt 'affichez', et immédiatement après une pression sur la touche RETURN, nous voyons que quelque chose s'affiche, à savoir:

```
5
```

ainsi que le terme:

```
Ok
```

Il sera évident que 5 est le résultat de l'addition indiquée 2+3. Le terme OK indique que l'ordinateur a achevé sa tâche et que nous pouvons lui en confier une nouvelle.

*Remarques:*
- Sur l'ordinateur nous ne pouvons absolument pas taper 2+3= (et ensuite appuyer sur la touche RETURN). Si nous le faisons malgré tout, il apparaît un message - en anglais - indiquant que nous avons fait une erreur.
- Avec l'ordinateur, si nous désirons voir immédiatement le résultat sur l'écran, on doit commencer par le terme PRINT.

**Multiplication et division**

Pour multiplier, on utilise le signe * et pour diviser on emploie la barre oblique. Ainsi, avec PRINT 5*36 on obtient 180 tandis que PRINT 180/36 donne 5 comme résultat.

**Nota** Le texte à afficher est toujours placé entre guillemets. Par exemple:

```
PRINT "CECI EST UN EXEMPLE"
```

donne comme résultat:

```
CECI EST UN EXEMPLE
```

Remarquez que les guillemets n'apparaissent pas dans le résultat. Nous passerons en revue d'autres possibilités un peu plus loin.

**Parenthèses**

Nous pouvons utiliser les parenthèses comme nous l'avons appris à l'école; ainsi

$$\frac{5+15}{23\ (17+103)}$$

est entré comme suit:

```
PRINT (5+15)/(23*(17+103))
```

Il faut remarquer que l'expression mathematique est placée ici sur une seule ligne. Le numérateur et le dénominateur ont été placés entre parenthèses. Si nous ne l'avions pas fait, l'instruction PRINT aurait été la suivante:

```
PRINT 5+15/23*(17+103)
```

Cette instruction mène à un résultat différent de celui qui a été demandé (ceci sera expliqué lorsque nous aborderons les règles de priorité des opérations).

Nous remarquons également qu'un signe de multiplication est placé dans le dénominateur entre 23 et 'ouvrir les parenthèses', puisqu'il s'agit dans l'expression originale du dénominateur '23(17+103)' d'une multiplication de 23 par (17+103).

**Ordre dans lequel les opérations sont exécutées**

Nous avons appris que la multiplication précède l'addition ou, autrement dit, la multiplication a une plus haute priorité que l'addition. Dans MSX-BASIC, les règles de priorité suivantes sont respectées:

1. **les expressions entre parenthèses sont d'abord exécutées;**
2. la multiplication et la division ont la même priorité, mais précèdent l'addition et la soustraction qui ont une priorité égale;
3. les opérations avec la même priorité sont exécutées de gauche à droite.
   Par exemple:

```
PRINT 15/3*5
```

devient 5*5

et donc 25

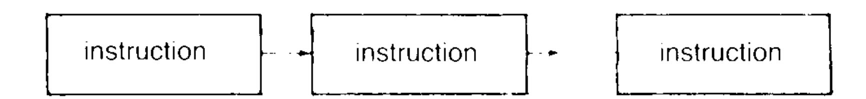
Donc, le résultat est 25 (et non pas 1!).
L'annexe F donne un aperçu complet des règles de priorité.

# 2 ET MAINTENANT UN PROGRAMME BASIC!

**Un programme BASIC: utilisation des numéros de ligne**

Un programme n'est rien d'autre qu'une série d'instructions qui, après avoir été introduites, sont exécutées automatiquement par l'ordinateur.

De manière schématique:




Dans le cas le plus simple nous pouvons introduire le programme à l'aide du clavier, comme s'il s'agissait d'un texte. Cela implique qu'un tel programme, ou série d'instructions, est stocké dans la mémoire.

Comment se fait-il que les instructions constituant le programme soient stockées dans la mémoire de l'ordinateur et ne soient pas exécutées immédiatement? La réponse est simple:

car il y a des numéros devant les instructions.

**Exemple**

Supposons que nous voulions laisser exécuter successivement par l'ordinateur les 3 instructions suivantes:

```
PRINT "CECI EST"
PRINT "LE PREMIER EXEMPLE"
PRINT "D'UN PROGRAMME"
```

Puisqu'il s'agit d'une *série* d'instructions, nous devons numéroter chaque instruction quand nous introduisons ce programme. Nous numérotons habituellement par 10, 20, 30, etc.
Donc nous introduisons:

```
10 PRINT "CECI EST"
20 PRINT "LE PREMIER EXEMPLE
30 PRINT "D'UN PROGRAMME"
```

A la fin de chaque ligne nous appuyons sur la touche RETURN, pour indiquer que la ligne a été introduite complètement.
(Attention: il ne faut pas oublier d'appuyer sur RETURN aussi après la dernière ligne!).
Quand toutes les lignes ont été introduites de cette façon, le programme dans son ensemble a été stocké dans la mémoire.

**Nous allons exécuter le programme**

Si nous introduisons maintenant:

RUN (suivie de la touche RETURN)

il apparaît sur l'écran:

```
CECI EST
LE PREMIER EXEMPLE
D'UN PROGRAMME
```

Hourra, nous avons exécuté un programme sur l'ordinateu

Dans ce cas, le programme consistait seulement en des structions PRINT pour imprimer le texte, mais néanmoin s'agissait d'une série d'instructions, autrement dit: d'un p gramme.

Nous faisons ici les remarques suivantes:

- En general quand un programme a été exécuté ou lan comme disent les experts - il est toujours conservé dans la moire.
  Seule une commande externe d'effacement, ou une coup d'alimentation de l'ordinateur, efface le programme de la moire.
- Les numéros indiquent également l'ordre dans lequel les structions sont exécutées, et ceci à partir du numéro le plus p jusqu'au numéro le plus élevé. Nous aurions pu entrer:

```
10 PRINT "CECI EST"
30 PRINT "D'UN PROGRAMME"
20 PRINT "LE PREMIER EXEMPLE"
```

  Le résultat aurait été le même, comme les numéros de ligne diquent. D'ailleurs, l'ordinateur met les instructions dans l dre décroissant dès qu'elles sont introduites.
- Nous ne pouvons utiliser comme numéro de ligne que nombres entiers (de 0 à 65529).
- Le fait de numéroter par 10, 20, 30 etc., donne l'avantage pouvoir insérer plus tard d'autres instructions.
- RUN est une commande, c'est-à-dire que l'ordinateur doit fectuer immédiatement l'action indiquée dès que la comma est entrée au clavier.

Autres commandes élémentaires:

LIST pour afficher le programme. A titre de démonstration, vez LIST et appuyez ensuite sur la touche RETURN.

AUTO pour générer automatiquement des numéros de ligne.

RENUM pour renuméroter les lignes

DELETE pour effacer l'une ou l'autre partie du programme.

NEW pour effacer la mémoire.

Ces commandes sont décrites de manière détaillée dans l'a çu des commandes MSX-BASIC. Essayez-les toutes à t d'exercice!

**Ajouter, insérer et modifier des lignes**

Dans la pratique, il est souvent necessaire de modifier un programme. Tout d'abord: efface le programme précédent au moyen de le commande NEW, puis entrez le programme qui suit:

```
10 PRINT "CECI EST UN"
20 PRINT "PROGRAMME"
30 PRINT "POUR ILLUSTRER"
```

**Ajouter**

Ajouter des lignes est très simple. Choisissez le numéro de ligne qui suit le dernier numéro. Par exemple:

```
40 PRINT "LE TRAVAIL"
50 PRINT "AVEC DES NUMEROS DE LIGNE"
```

Entrez la commande LIST pour constater que ces lignes ont effectivement été ajoutées.

**Insérer**

Pour insérer une ligne, on choisit un numéro de ligne compris entre le numéro des lignes entre lesquelles la nouvelle ligne doit figurer. Par exemple:

```
25 PRINT "COURT ET SIMPLE"
```

Vérifiez avec la commande LIST que cette ligne se trouve à la place souhaitée après avoir été introduite au moyen du clavier.

**Effacer**

Pour effacer une ligne, il suffit d'écrire le numéro de la ligne à supprimer et d'appuyer sur la touche RETURN. Par exemple:

```
25 (touche RETURN)
```

Vérifiez à nouveau au moyen de la commande LIST. Eventuellement, on peut aussi utiliser la commande DELETE (voir l'aperçu des commandes MSX-BASIC).

**Correction d'erreurs**

Nous avons déjà vu ci-dessus quelques possibilités pour modifier un programme. Dans ce paragraphe, nous montrons ce qui peut encore être fait.

La solution apparemment la plus simple est de retaper la ligne comportant l'erreur, en la corrigeant.

Introduisez maintenant:

```
25 PRINT "COURD ET SIMPLE"
```

Après la commande LIST, nous voyons comment cette ligne comportant une erreur évidente, a été incluse dans le programme.

La solution la plus élégante est obtenue au moyen des tou-

ches de commande du curseur. Ce sont les touches munies de flèches à droite du clavier. Appuyez sur l'une de ces touches et voyez comment le curseur se déplace à l'écran.

Ce petit carré est appelé curseur, et c'est pourquoi les touches avec flèches sont appelées les touches de commande du curseur.

Maintenant, déplacez le curseur de telle façon qu'il se positionne précisément sur l'erreur, donc

```
25 PRINT "COUR[D] ET SIMPLE"
               ↑
```

*indique le curseur*

Ensuite appuyez sur T et la touche RETURN, et voilà l'erreur corrigée!

Si l'on maintient la touche curseur enfoncée, tout se passe comme si cette touche était enfoncée á plusieurs reprises. Cet effet d'auto-répétition s'applique également aux autres touches.

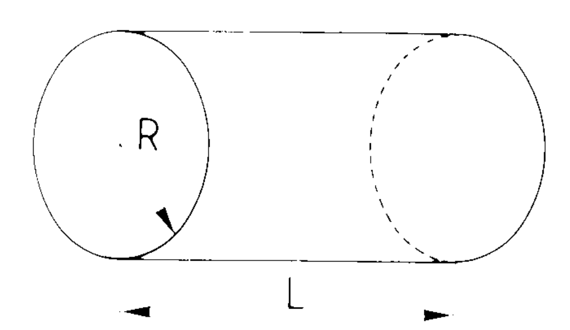
**Touches spéciales**

Enfin, remarquons qu'il est possible d'effacer un texte en appuyant sur la touche BS. Avec la touche INS, nous pouvons insérer des caractères, et avec la touche SUP, nous pouvons en effacer. Ces deux dernières touches sont importantes surtout quand nous travaillons avec les touches du curseur. D'abord, nous déplaçons le curseur vers l'erreur et ensuite nous pouvons, à volonté, modifier, effacer ou insérer des caractères.

# 3 VARIABLES ET ARITHMÉTIQUE

**Introduction**

Pour déterminer le volume d'un cylindre, nous avons besoin de connaître la surface de la section et nous devons multiplier cette valeur par la longueur (L).




surface de la section = 3.14159xRxR

volume du cylindre = surface de la section x L = 3.14159xRxRxL

Nous voyons comment le volume d'un cylindre dépend de deux mesures: le rayon R et la longueur L.

Le programme suivant calcule le volume d'un cylindre et ceci pour les valeurs suivantes:
L=5
et R=7

Programme:

```
10 L=5
20 R=7
30 PI=3.14159
40 SUR=PI*R*R
50 I=SUR*L
60 PRINT I
```

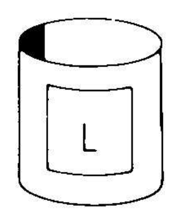
Après la commande RUN le programme répond:

```
769.68955
```

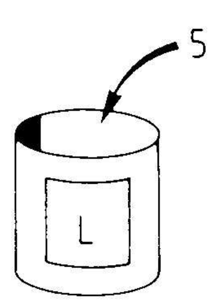
Examinons ce programme ligne par ligne. A la ligne 10 nous avons l'expression:

L=5

Cette expression indique à l'ordinateur de réserver une partie de sa mémoire et y associe un nom L. Un tel emplacement en mémoire peut être considéré comme une boîte.

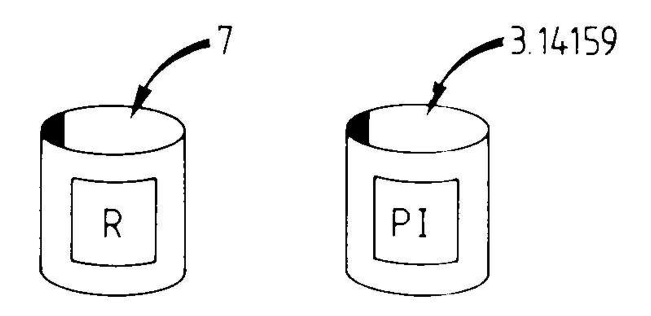



L'expression L=5 veut dire alors que la valeur 5 est stockée dans cette boîte.




Quand le programme fait référence à L, l'ordinateur regardera en effet le *contenu* de cette boîte.

Les lignes 20 et 30 peuvent être visualisées de la même façon:




Remarquez que, dans PI, le nombre 3.14159 est stocké et non pas 3,14159. La virgule que nous utilisons habituellement est toujours indiquée par un point en MSX-BASIC. Dans le résultat du programme nous voyons également ce point décimal.

Ceux qui ont quelques connaissances mathématiques auront reconnu dans le nombre 3.14159 le nombre $\pi$ (prononcez: pi).

La ligne 40 indique une opération dans la boîte portant le nom SUR sera stockée la valeur obtenue en multipliant PI par RxR.
La valeur 3.14159x7x7 sera donc stockée dans SUR.
La valeur ainsi trouvée est utilisée dans la ligne 50
pour déterminer finalement la valeur du contenu. Celui-ci est stocké dans la boîte portant l'étiquette I.
Finalement, nous utilisons de nouveau une instruction
PRINT pour afficher le contenu de cette dernière boîte.

```
PRINT I
```

↗ *affichez* — ↖ *le contenu de la boîte avec l'étiquette I*

Ce n'est pas la lettre I qui est affichée ici, pour cela, l'instruction aurait dû être PRINT "I", autrement dit: nous aurions dû placer I entre guillemets.

Ce premier programme appelle les remarques suivantes:

- Au lieu de:

  ```
  10 L=5
  ```

  nous aurions pu écrire aussi:

  ```
  10 LET L=5
  ```

  Ce qui veut dire littéralement 'Soit L égal à 5'.

  Avec cette expression nous insistons sur le fait que l'ordinateur exécute une *action* : quelque chose est mis dans la boîte avec l'étiquette L. Cette action devient tout à fait claire dans l'exemple suivant:

  ```
  10 LET X=3
  20 LET X=X+4
  ```

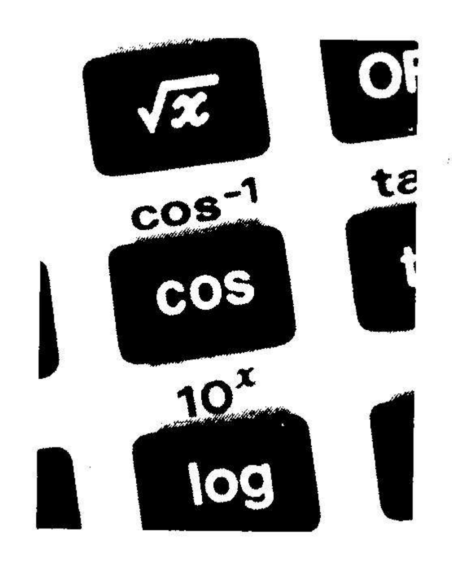
  A la deuxième ligne, 4 est ajouté à l'ancienne valeur de X, donc 3+4, est mise dans X. Ainsi X comprendra finalement la valeur 7. Attention: une expression comme X=X+4 est parfaitement correcte en BASIC, tandis qu'elle n'est pas permise en mathématique.

- Le fait que nous puissions mettre n'importe quelle valeur dans une boîte, indique que le contenu est variable; c'est la raison pour laquelle nous parlons habituellement d'une *variable* au lieu d'une boîte. Ainsi notre programme connaît les variables L, R, PI, SUR et I. Dorénavant nous parlerons donc 'd'attribuer une variable' au lieu de 'stocker une valeur dans une boîte'.

- Certaines règles s'appliquent à la dénomination des variables:
  - Seules les deux premières lettres d'un nom sont reconnues par l'ordinateur. Ainsi, les mots UNIQUE et UNE sont identiques pour l'ordinateur: seul UN est reconnu.
  - On ne peut pas utiliser de mots réservés comme noms; ainsi, on ne peut pas utiliser IF comme nom, puisque c'est un mot réservé. Les mots réservés sont donnés dans l'annexe K.

**Fonctions standards**

L'exemple ci-dessus est encore trop simple. Pour imaginer que nous pouvons faire des calculs compliqués de cette façon. L'étude des fonctions standards approfondira considérable-




ment la compréhension de ce que nous pouvons calculer avec MSX-BASIC.

Pour expliquer les fonctions standards, regardons d'abord une calculatrice, plus précisément une calculatrice pourvue de fonctions mathématiques. Quand nous devons calculer la racine d'un nombre, avec une telle calculatrice nous introduisons un nombre et, ensuite, nous appuyons sur la touche $\sqrt{\ }$. Ainsi, la fonction désirée est effectuée simplement en enfonçant la touche prévue à cet effet.

Avec MSX-BASIC nous pouvons également calculer d'une manière simple des valeurs de fonction. Une fonction de ce type est désignée par un certain terme et on met entre parenthèses la valeur de la fonction à calculer, soit:

fonction (valeur)

↗ *indique la fonction* ↖ *indique la valeur de la fonction*

Exemples:

```
10 A=SQR(4)
20 PRINT A
```

Résultat:

```
2
```

A la ligne 10, la racine de 4 est attribuée à A. Le terme qui, dans cet exemple, indique la fonction est SQR. Entre parenthèses se trouve la valeur dont la racine doit être déterminée, dans ce cas (par simplicité) 4.

Nous remarquons que nous pouvons également avoir entre parenthèses une variable, ou même une expression arithmétique:

Exemples:

| | | |
|---|---|---|
| `B=SQR(A)` | : | ici la valeur est indiquée par la variable A. |
| `C=SQR(SQR(5)+4+A)` | : | ici la valeur est indiquée par l'expression SQR(5)+4+A. Remarquez que cette expression elle-même comporte une fonction. |

Le Tableau suivant donne les fonctions les plus connues.

L'aperçu des commandes MSX-BASIC passe toutes les fonctions en revue.

| Désignation MSX-BASIC | Notation algébrique | Signification |
|---|---|---|
| ABS (X) | x | Donne la valeur absolue de X. C'est la valeur 'sans signe'. |
| ATN (X) | arctan (x) | Donne l'arctangente de X. (résultat en radians entre –pi/2 et pi/2). |
| COS (X) | cos (x) | Donne le cosinus de X (X en radians). |
| EXP (X) | $e^x$ | Donne la puissance X de e. |
| INT (X) | aucune | Arrondit toujours la valeur de X au chiffre inférieur. Ainsi INT (3.8) donne la valeur 3 et INT (-3.1) la valeur -4. Nous pouvons arrondir un nombre à la valeur entière la plus proche avec INT (X + 0.5). |
| LOG (X) | ln (x) | Donne le logarithme naturel de X. |
| SGN (X) | aucune | Donne la valeur -1, 0, ou 1 tenant compte si X est négatif, 0 ou positif. |
| SIN (X) | sin (x) | Donne le sinus de X: (X en radians). |
| SQR (X) | $\sqrt{x}$ | Donne la racine de X. |
| TAN (X) | tan (x) | Donne la tangente de X: (X en radians). |

Nous faisons remarquer que l'élévation à une puissance ne figure pas dans cette table. Pour cela nous utilisons le signe ∧ . Ainsi PRINT 2 ∧ 3 donne la valeur 8 comme résultat ($=2^3$).

**Nota** Pour obtenir le signe 'd'élévation à la puissance' il faut taper sur la touche de l'accent circonflexe, puis sur la barre d'espace.

**Plusieurs instructions sur une seule ligne**

Le MSX-BASIC offre également la possibilité de mettre plusieurs instructions sur une même ligne.

On utilise le signe ':' ('deux points') comme signe de séparation. Voici un programme à titre d'illustration..

*sans signe de séparation*

```
10 A=10
20 B=5
30 C=A+B
40 PRINT C
```

*avec signe de séparation*

```
10 A=10:B=5:C=A+B:PRINT C
```

# 4 INPUT, READ ET DATA

**INPUT** La première instruction étudiée est l'instruction INPUT. Celle-ci nous offre la possibilité d'attribuer des valeurs à des variables lors de l'exécution d'un programme. Ce qui est peut-être le plus important, c'est que cette instruction offre la possibilité d'écrire des programmes à usage général. Tout d'abord, regardons une fois de plus le programme du chapitre précédent:

```
10 L=5
20 R=7
30 PI=3.14159
40 SUR=PI*R*R
50 I=SUR*L
60 PRINT I
```

Il va sans dire que nous aurions pu également introduire:

```
PRINT 3.14159*7*7*5
```

pour obtenir le même résultat.

Nous ne l'avons pas fait, parce que nous avons voulu montrer comment nous pouvons stocker dans la mémoire de l'ordinateur un schéma de calcul avec lequel nous n'avons qu'à remplir les valeurs de R et de L pour déterminer le résultat.

Pour déterminer par exemple le volume d'un cylindre avec *L* 34 et *R* 10, nous pouvons modifier les lignes 10 et 20:

```
10 L=34
20 R=10
```

et après la commande RUN, la réponse désirée apparaît. Seulement, la modification d'un programme pour exécuter un calcul est à éviter, puisque nous pourrions faire des erreurs...

Le programme serait bien meilleur si, après la commande RUN, des messages apparaissaient sur l'écran indiquant qu'il faut introduire des valeurs pour *L* et *R* .

Cette possibilité est offerte par l'instruction INPUT:

```
INPUT "texte à afficher"; nom de la variable
```

Par exemple:

```
INPUT "DONNEZ LA LONGUEUR"; L
```

Le résultat de cette instruction est que l'ordinateur affiche le texte, y compris un point d'interrogation, et interrompt le programme immédiatement après, attendant que l'utilisateur ait introduit un nombre, par exemple:

```
5    (et la touche RETURN)
```

Dès que la touche RETURN est enfoncée, le nombre est attribué la variable *L*, et l'ordinateur reprend l'exécution du programme.

Avec les instructions INPUT, notre programme devient le suivant:

```
10 INPUT "DONNEZ LA LONGUEUR"; L
20 INPUT "DONNEZ LE RAYON"; R
30 PI=3.14159
40 SUR-PI*R*R
50 I SUR*L
60 PRINT "VOLUME-"; I
```

Le résultat est par exemple:

```
DONNEZ LA LONGUEUR? 5
DONNEZ LE RAYON? 7
VOLUME- 769.68955
```

On voit comment, avec l'instruction PRINT, un texte peut être inclu dans un programme, en mettant ce texte entre guillemets et en utilisant le signe ';' ('point-virgule') de séparation avec la variable.

Le programme a été considérablement amélioré par rapport au premier programme. Sa structure est entièrement déterminée. Nous n'avons pas besoin d'effectuer des modifications dans le programme pour calculer le volume de *n'importe quel* cylindre.

**Remarques** L'instruction INPUT appelle les remarques suivantes:

- Nous pouvons introduire plusieurs variables dans une instruction INPUT:

```
INPUT "INTRODUISEZ A, B ET C:"; A,B,C
```

Après l'interruption du programme, l'ordinateur affiche le texte et un point d'interrogation, et nous devons introduire les trois nombres, séparés par une virgule.

- Nous pouvons également supprimer le texte dans l'instruction INPUT, par exemple:

```
INPUT A
```

Après l'interruption, l'ordinateur ne montre que le point d'interrogation pour indiquer que nous devons introduire un nombre.

**READ et DATA** La troisième possibilité d'attribuer des valeurs, que nous abordons ici, répond au besoin d'attribuer des valeurs à une série relativement longue de variables. En général, ces valeurs ne peuvent subir de modifications d'un programme à un autre.

Evidemment, nous pourrions incorporer toute une série d'instructions LET dans le cas présent, mais READ et DATA sont des solutions beaucoup plus élégantes comme nous le verrons plus loin. Voici tout de suite un exemple:

```
10 READ A,B,C,D
20 F=A+B+C+D
30 PRINT F
40 DATA 3,7,4,8
```

READ, en ligne 10, indique à l'ordinateur qu'un nombre de variables est mentionné, et que leurs valeurs sont énumérées dans la ligne commençant par DATA.

Ainsi, la valeur 3 sera attribuée à A, la valeur 7 à B, la valeur 4 à C et la valeur 8 à D.
Le résultat du programme le confirme:

```
22
```

Remarquez que la ligne DATA ne constitue pas une instruction exécutable. Ce n'est qu'une liste dans laquelle l'ordinateur trouve les valeurs recherchées. D'ailleurs, nous aurions également pu diviser la ligne READ:

```
10 READ A,B
15 READ C,D
```

de même que nous aurions pu diviser la liste DATA, par exemple de la façon suivante:

```
40 DATA 3,7,4
50 DATA 8
```

Pour l'ordinateur, tout cela ne fait aucune différence; il lit les données comme si elles formaient une seule liste continue. On peut imaginer le mécanisme suivant: l'ordinateur met d'abord une flèche à la première valeur de la liste DATA. Chaque fois qu'une valeur est attribuée à une variable au moyen d'une instruction READ, cette flèche avance d'une position:

```
40 DATA 3,7,4,8
          ↑
```

*la flèche avance d'une position à chaque lecture*

Quand la dernière valeur de la liste DATA a été atteinte, l'ordinateur pointe la flèche vers la première valeur d'une liste DATA suivante si, du moins, il y en a une.

Nous pouvons également remettre cette flèche en position originelle, au moyen de l'instruction suivante:

```
RESTORE
```

Par exemple:

```
10 READ A,B
20 RESTORE
30 READ C,D
40 F=A+B+C+D
50 PRINT F
60 DATA 3,7,4,8
```

Résultat:

```
20
```

Remarquez que le résultat est maintenant 20. C'est la conséquence de l'instruction RESTORE; les valeurs 3 et 7 sont attribuées à A et B, aussi bien qu'à C et D.

# 5 NOMBRES, GRANDS ET PETITS, ET LEURS REPRÉSENTATIONS

**Introduction** Avant d'aller plus loin, examinons le tableau reproduit ci-dessous.

| *intitulé* | *notation* | *signification* | *valeur* |
|---|---|---|---|
| 10 à la puissance 1 | $10^1$ | 10 | 10 |
| 10 à la puissance 2 | $10^2$ | 10x10 | 100 |
| 10 à la puissance 3 | $10^3$ | 10x10x10 | 1000 |
| 10 à la puissance 4 | $10^4$ | 10x10x10x10 | 10000 |
| 10 à la puissance -1 | $10^{-1}$ | 1/10 | 0.1 |
| 10 à la puissance -2 | $10^{-2}$ | 1/(10x10) | 0.01 |
| 10 à la puissance -3 | $10^{-3}$ | 1/(10x10x10) | 0.001 |
| 10 à la puissance 0 | $10^0$ | 10/10 | 1 |

On peut en conclure que $10^4$ correspond à 10000, autrement dit: le nombre de zéros correspond ici directement à la puissance (le chiffre à droite en haut du 10).

*Exemple*:

A quelle puissance correspond 1 000 000 000?

Réponse: comptez le nombre de zéros. Il y en a 9 et donc nous trouvons $10^9$.

Le tableau montre également que 0.001 correspond à $10^{-3}$. Ainsi, avec les nombres inférieurs à 1 nous pouvons apparemment compter le nombre de zéro pour déterminer la puissance.

*Exemple*:

A quelle puissance correspond le nombre suivant?

0.000 000 000 000 000 000 1

Réponse: comptez le nombre de zéros. Il y en a 19 et donc nous trouvons $10^{-19}$.

Ce nombre peut également s'écrire $1 \times 10^{-19}$

le premier nombre est appelé la *mantisse* et le deuxième l'*exposant* .

Avec l'ordinateur, nous indiquons le nombre précédent avec la notation suivante:

1E-19

Autrement dit, nous utilisons la lettre E pour distinguer la mantisse de l'exposant.

Pour acquérir un peu d'expérience dans cette notation, nous pouvons utiliser le programme suivant:

```
10 INPUT A
20 INPUT B
30 C=A*B
40 PRINT C
```

*Exemples:*

| | | |
|---|---|---|
| A=987654 | B=456789 | donne: 451149483006 |
| A=98765434 | B=987654321 | donne: 9.754610765554E+16 |
| A=.0000000008 | B=.0000000007 | donne: 5.6E-19 |
| A=8000000000 | B=7000000000 | donne: 5.6E+19 |

**Simple et double précision**

Pour la représentation de nombres, notre ordinateur utilise un nombre limité de registres de mémoire (mots). Cela veut dire également que les calculs ont une précision limitée. MSX-BASIC offre à son utilisateur la possibilité de travailler aussi bien avec la double précision qu'avec la simple précision. Avec la double précision, 8 registres sont toujours utilisés pour le stockage de nombres, tandis que la simple précision n'en utilise que 4. Notre ordinateur travaille normalement avec des nombres en double précision. Pour travailler avec des nombres en simple précision, mettez toujours un point d'exclamation derrière le nombre.

Les variables auxquelles nous attribuons des nombres en simple précision, se distinguent des variables en double précision par la mise de ! directement derrière le nombre. Les exemples suivants illustrent la différence entre les calculs en double précision et en simple précision.

```
10 A=10/3
20 PRINT A
```

résultat

3.3333333333333

```
10 A!=10!/3!
20 PRINT A!
```

résultat

3.33333

Faisons ici les remarques suivantes:

- Avec les nombres en double précision, nous pouvons mettre à la fin du nombre le signe #. Nous pouvons donner également ce signe comme dernier signe d'un nom. Ainsi, nous indiquons de manière explicite que nous travaillons avec des nombres en double précision.
- Quand nous utilisons la lettre D au lieu de la lettre E (p.e. 23.45156321D39), nous indiquons de manière explicite que nous travaillons en double précision.
- Il faut toujours se demander s'il est sage de représenter les résultats avec autant de chiffres. Imaginons qu'un charpentier doive scier une planche de 10 mètres en 3 parties. Chaque partie aura selon notre ordinateur une longueur de 3.333 333 333 333 3 m. Mais notre charpentier saura-t-il scier avec une telle précision?

- L'emploi de nombres en simple précision peut être important surtout quand nous voulons économiser la capacité mémoire. Ce problème de capacité mémoire se révèle surtout, en pratique, avec les tableaux.

**Nombres entiers**

Dans un certain nombre de cas, nous voulons indiquer dans le programme qu'il s'agit absolument de nombres entiers (en anglais: integer).

Pour la représentation de nombres dans un programme cela ne pose pas de problèmes, puisqu'un nombre sans point décimal est un nombre entier. Cependant, on ne peut voir si des variables représentent un nombre entier ou un nombre décimal. Or, pour rendre cette distinction évidente, nous mettons le signe % après le nom.

Exemples:

```
A%      B1%
```

Attention, une telle variable ne peut comporter qu'un nombre entier. Si le résultat d'un calcul donne un nombre décimal celui-ci est arrondi automatiquement à l'entier inférieur.

Exemple:

```
10 A%=20
20 B%=3
30 C%=A%/B%
40 PRINT C%
```

Résultat:

```
6
```

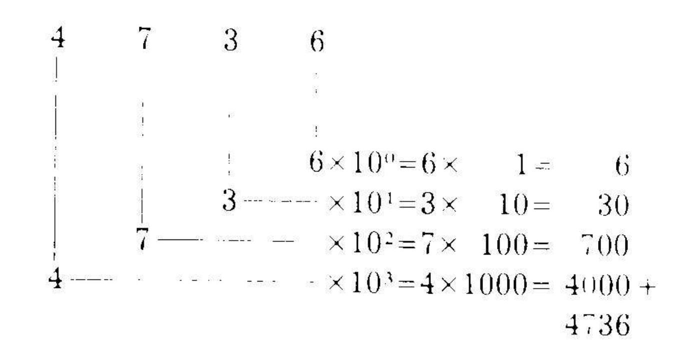
Remarquez, une fois de plus, que l'arrondi a toujours lieu au chiffre inférieur. Dans notre exemple 6.6666.. a été arrondi à 6!

**Nombres binaires**

Pour expliquer les nombres binaires, regardons d'abord la structure des nombres 'ordinaires', c'est-à-dire décimaux. Ces nombres consistent toujours en des puissances de 10 (voir la table au début de ce chapitre).

Exemple:

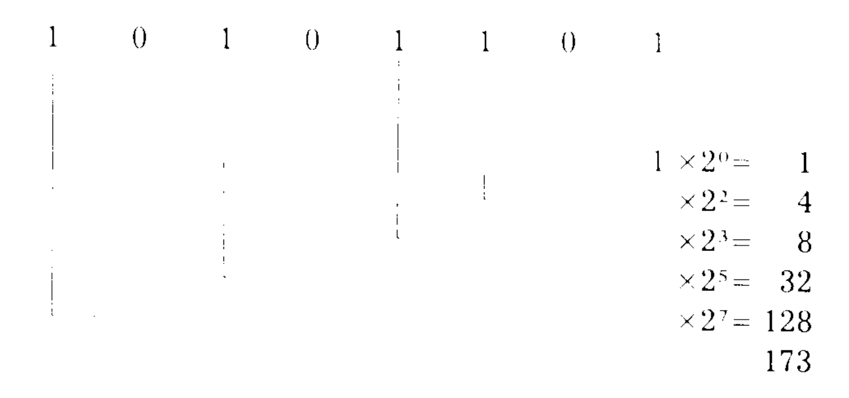
Le nombre 4736 est formé de puissances de 10 de la manière suivante:



Nous voyons comment chaque chiffre correspond ici à une puissance de 10. En partant de la droite, nous comptons les positions en commençant par 0, 1, 2 et 3; celles-ci correspondent alors exactement aux puissances de 10.

Or, les nombres binaires ne sont formés qu'avec les chiffres 0 et 1 et la position d'un tel chiffre correspond à une puissance, à savoir une puissance de 2.
Exemple:

A quel nombre décimal correspond le nombre binaire 10101101?




En MSX-BASIC, nous pouvons aussi indiquer directement les nombres binaires, en mettant &B devant le nombre.
Exemple:

```
10 A%=&B10101101
20 PRINT A%
```

Résultat:

```
173
```

## Nombres octaux et hexadécimaux

Nous pouvons maintenant étudier d'une manière simple ce que sont les nombres octaux et hexadécimaux.

Les nombres octaux sont des nombres dont le chiffre correspond selon notre explication à une puissance de 8. Dans les nombres hexadécimaux, la position d'un chiffre correspond à une puissance de 16.

Il est intéressant de remarquer que nous avons exactement 10 chiffres (0 à 9) pour représenter les nombres décimaux. Dans les nombres binaires, il n'y a que deux chiffres (0 et 1) et dans les nombres octaux, il n'y en a évidemment que 8 (0 à 7).

Aucun problème jusqu'à maintenant. Dans les nombres hexadécimaux il y a naturellement 16 chiffres. Cependant, nous ne connaissons que les chiffres 0 à 9. Comment indiquer alors les autres nombres hexadécimaux? Nous utilisons à cette fin les lettres A à F.

La table suivante donne les nombres décimaux de 0 à 15, qui

sont également donnés, à titre d'exemple, dans leurs formes binaire, octale et hexadécimale.

| *décimal* | *binaire* | *octal* | *hexadécimal* |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 |
| 2 | 10 | 2 | 2 |
| 3 | 11 | 3 | 3 |
| 4 | 100 | 4 | 4 |
| 5 | 101 | 5 | 5 |
| 6 | 110 | 6 | 6 |
| 7 | 111 | 7 | 7 |
| 8 | 1000 | 10 | 8 |
| 9 | 1001 | 11 | 9 |
| 10 | 1010 | 12 | A |
| 11 | 1011 | 13 | B |
| 12 | 1100 | 14 | C |
| 13 | 1101 | 15 | D |
| 14 | 1110 | 16 | E |
| 15 | 1111 | 17 | F |

Pour le moment l'utilité de tout cela semble être encore très lointaine. Pourtant, nous verrons (entre autres dans le chapitre sur les sprites), comme tout cela sera très utile.

Les nombres octaux et hexadécimaux peuvent être introduits directement tels quels dans les programmes. Devant un nombre octal, nous mettons alors &O et devant un nombre hexadécimal, &H.

Exemples:

```
PRINT &O17  donne  15
PRINT &HF   donne  15
PRINT &HFF  donne  255
```

# 6 PRINT, TAB, LOCATE, PRINT USING ET REM

Dans les chapitres précédents, nous avons vu que nous pouvions imprimer plusieurs choses avec l'instruction PRINT.

Prenons un exemple:

```
10 A=12
20 PRINT "A=";A
30 PRINT "A=",A
```

Résultat:

```
A=12
A=                     12
```

La différence entre le résultat de la ligne 20 et celui de la ligne 30 vient des signes de séparation. En ligne 20, nous utilisons le point-virgule comme signe de séparation et en ligne 30, la virgule.

Avec un point-virgule, l'impression suivante se place immédiatement après l'impression précédente.

Quand nous utilisons une virgule, l'ordinateur effectue automatiquement une division en colonnes. Dans cette division en colonnes, il y a toujours un espace intermédiaire de 14 positions.

Ces signes de séparation peuvent également être mis à la fin de la commande PRINT, comme le montre l'exemple suivant:

```
10 PRINT "ABC";"DEFG";
20 PRINT "H";"IJ";"KLM"
```

Résultat:

```
ABCDEFGHIJKLM
```

Quand nous désirons sauter une ligne dans l'impression d'un texte, nous utilisons une instruction PRINT sans autre spécification.
Exemple:

```
10 PRINT "A"
20 PRINT
30 PRINT "B"
```

Résultat:

```
A

B
```

Il est évident qu'une ligne a été sautée entre A et B.

**TAB**

Au moyen de la commande TAB, on peut indiquer à partir de quelle colonne un texte à afficher doit commencer.
L'exemple suivant explique l'emploi de TAB.

```
10 PRINT TAB(1); "MSX"
20 PRINT TAB(3); "MSX"
30 PRINT TAB(5); "MSX"
```

Résultat:

```
MSX
  MSX
    MSX
```

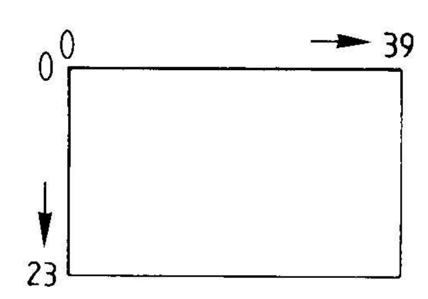
Le positionnement en colonne, de ce qui est à imprimer, est indiqué par le nombre entre parenthèses situé après TAB.

**LOCATE**

La fonction TAB règle la position d'impression sur la ligne. LOCATE indique la position horizontale *et* la position verticale. C'est une instruction importante qui permet d'obtenir d'une façon simple une belle présentation du texte à imprimer.
La forme de l'instruction LOCATE est la suivante:

`LOCATE` position horizontale, position verticale

La disposition de l'écran est alors:




Exemple de programme:

```
10 CLS
20 LOCATE 20,10
30 PRINT "MSX"
```

Résultat:

Le programme commence par une instruction que nous n'avons pas encore traité, à savoir l'instruction CLS (CLear Screen = effacez l'écran). Celle-ci a pour résultat d'effacer l'écran et de positionner le curseur dans le coin gauche supérieur. L'instruction LOCATE qui suit déplace le curseur vers la position (20,10). Le texte MSX est alors affiché à partir de cette position.

**WIDTH**

A ce sujet, remarquons également que l'instruction WIDTH permet de fixer le nombre de positions horizontales. Pour une description complète, voyez les instructions SCREEN et WIDTH dans l'aperçu des commandes MSX-BASIC.

**PRINT USING** Dans de nombreuses applications, nous désirons que les résultats apparaissent toujours sous une certaine forme. Les calculs financiers en constituent un exemple. Leurs résultats doivent apparaître toujours sous une forme donnée avec deux chiffres après la virgule.

La façon dont nous affichons un nombre est indiquée par l'instruction PRINT USING.

Celle-ci a toujours la structure suivante:

PRINT USING "information sur représentation"; nombre

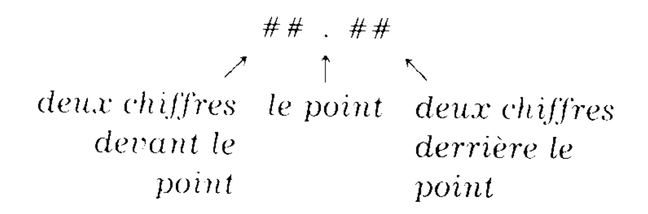
Voici un exemple:

```
PRINT USING "##.##"; 32.7
```

L'écran donne alors:

```
32.70
```

L'expression entre guillemets, autrement dit ##.## indique comment le nombre doit être représenté. Dans ce cas il s'agit d'un chiffre à deux décimales.




les possibilités d'emploi de PRINT USING sont très nombreuses.

On en trouvera une description complete dans l'aperçu des commandes MSX-BASIC.

**REM** La dernière instruction étudiée dans ce chapitre est l'instruction REM. Le terme REM est l'abréviation de 'remarque'. Cette instruction nous permet d'ajouter un commentaire à un programme. Ce commentaire doit alors être placé derrière le terme REM.

Un exemple:

```
10 REM UN EXEMPLE QUELCONQUE
20 PRINT "FIN"
```

Résultat après la commande RUN:

```
FIN
```

Nous voyons que le commentaire placé derrière REM n'est pas imprimé après la commande RUN. Nous le retrouvons seulement quand nous donnons à nouveau la commande LIST.

# 7 LES INSTRUCTIONS OPÉRATOIRES: GOTO, IF...THEN, FOR...NEXT ET ON...GOTO

**GOTO** L'instruction GOTO a une forme très simple, à savoir:

```
GOTO  numéro de ligne
```

Quand l'ordinateur rencontre cette instruction, un saut vers le numéro de ligne indiqué est fait, et le programme se poursuit alors à partir du numéro de ligne indiqué. Comme ce saut est *toujours* effectué, nous parlons ici de saut *inconditionnel* . Un exemple simple:




```
10 PRINT "CECI EST IMPRIME"
20 GOTO 40
30 PRINT "CECI PAS"
40 PRINT "CECI EST IMPRIME AUSSI"
```

Résultat:

```
CECI EST IMPRIME
CECI EST IMPRIME AUSSI
```

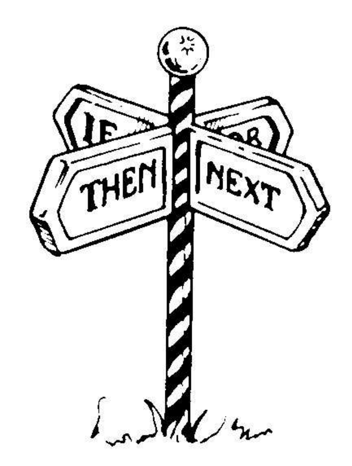
La ligne 20 donne un saut en la ligne 40, de sorte que la ligne 30 est toujours sautée.

**IF...THEN** La forme la plus simple de l'instruction IF...THEN est:

```
IF  condition  THEN  instruction
```

Nous voyons qu'entre les termes IF et THEN une condition est indiquée; si celle-ci (après IF) est remplie, l'instruction donnée après THEN (ALORS) sera exécutée. Nous pouvons expliquer la construction au moyen d'un exemple très simple:

```
10 INPUT "INTRODUISEZ NOMBRE" K
20 IF K=3 THEN PRINT "NOMBRE ETAIT TROIS"
30 PRINT "FIN DE PROGRAMME"
```




Le programme commence par une instruction INPUT qui indique l'introduction d'un nombre. Imaginons que nous introduisions la valeur 3; la valeur 3 est donc attribuée à K. Dans la ligne suivante il y a une instruction IF...THEN.
La condition est:

```
K=3
```

ou en clair:

est-ce que K est égal à 3?

Remarquez que nous avons mis l'expression sous la forme interrogative. Or, une telle expression ne peut être que vraie ou fausse; d'oú, dans notre exemple, K est égal à 3 ou il ne l'est pas.

Nous disons alors que la condition indiquée est *vraie* ou fausse. Autrement dit: l'expression logique est vraie ou est fausse. Dans le cas présent, la valeur 3 a été attribuée à K, et l'on en conclut que la condition posée a été remplie.

Dans ce cas, l'ordinateur exécutera l'instruction après THEN, avec pour résultat l'affichage du message 'LE NOMBRE ETAIT TROIS'. Si K n'était pas égal à 3, autrement dit si la condition posée n'était pas remplie, l'instruction suivant THEN serait simplement sautée. Quand l'ordinateur a exécuté l'instruction IF...THEN, il passe à l'instruction suivante.

Dans l'exemple suivant nous montrons un programme où THEN est suivi d'un GOTO:

```
10 PRINT "COMBIEN FONT 2+5?"
20 INPUT K
30 IF K=7 THEN GOTO 80
40 PRINT "MAIS NON"
50 PRINT "LA REPONSE A 2+5"
60 PRINT "EST NATURELLEMENT 7"
70 GOTO 90
80 PRINT "C'EST EN EFFET LA REPONSE"
90 END
```

Nous voyons ici qu'après THEN l'instruction GOTO 80 a été utilisée. Si K prend la valeur 7, un saut vers la ligne 80 est effectué.

Si une valeur différente de 7 est introduite pour K, cette instruction GOTO est sautée et l'ordinateur passe aux lignes 40, 50, 60 et 70. Remarquez que la ligne 70 comporte de nouveau une instruction GOTO. Celle-ci est nécessaire, sinon le texte de la ligne 80 apparaîtrait après le texte des lignes 40, 50 et 60, ce qui serait bizarre.

La dernière instruction du programme est l'instruction END, qui indique que le programme est terminé. Eventuellement cet instruction peut être supprimée. C'est d'ailleurs ce que nous avons fait dans les programmes précédents.

Ce programme appelle encore les remarques suivantes:

- Au lieu de `IF K=7 THEN GOTO 80`
  nous aurions pu écrire
  `IF K=7 THEN 80`
  ou avec suppression de THEN
  `IF K=7 GOTO 80`

- Après THEN, nous aurions pu mettre d'autres instructions, séparées par les signes ':'. La seule condition à remplir est que l'expression IF...THEN soit contenue en entier dans une seule ligne BASIC (255 caractères au maximum).

- Dans l'instruction IF...THEN, K est comparé à 7, et ceci au moyen du signe =. Cet opérateur est appelé opérateur relationnel. En plus du signe =. nous pouvons également utiliser les signes suivants:

| *signe* | *signification* |
|---|---|
| < | plus petit que |
| > | plus grand que |
| <= | inférieur ou égal à |
| =< | inférieur ou égal à |
| >= | supérieur ou égal à |
| => | supérieur ou égal à |
| <> | différent de |
| >< | différent de |

- Nous pouvons aussi combiner des expressions logiques. Ces possibilités sont discutées en Annexe F.

**IF...THEN ...ELSE**

MSX BASIC permet d'utiliser l'instruction IF...THEN avec ELSE. Nous expliquons cette extension au moyen d'un exemple simple:

```
10 INPUT K
20 IF K=7 THEN PRINT "K=7" ELSE
               PRINT "K N'EST PAS EGAL A 7"
30 END
```

L'instruction suivant THEN n'est exécutée que si la condition spécifiée (K=7?) est remplie. Dans tous les autres cas, l'instruction suivant ELSE sera exécutée. La encore, comme avec THEN, nous pouvons indiquer plusieurs instructions suivant ELSE, séparées par deux points.

**FOR...NEXT**

L'instruction FOR NEXT doit être considérée comme un instrument de programmation très facile, avec lequel nous pouvons indiquer qu'une certaine série d'instructions doit être exécutée à plusieurs reprises.
L'exemple suivant l'explique:

```
10 FOR A=1 TO 5
20 PRINT A; "CECI EST UNE DEMONSTRATION"
30 NEXT A
```

Résultat:

```
1 CECI EST UNE DEMONSTRATION
2 CECI EST UNE DEMONSTRATION
3 CECI EST UNE DEMONSTRATION
```

```
4 CECI EST UNE DEMONSTRATION
5 CECI EST UNE DEMONSTRATION
```

Les instructions qui doivent être exécutées à plusieurs reprises, sont toujours incluses entre la ligne commençant par FOR et la ligne finissant par NEXT, donc schématiquement:

```
.. FOR nom de la variable=...
.....  }
.....  } ces instructions sont exécutées à plusieurs reprises
.....  }
.. NEXT nom de la variable
```

Dans notre cas, il s'agit d'une seule instruction, à savoir celle de la ligne 20. On définit dans la ligne commençant par FOR combien de fois les instructions demandées doivent être exécutées.

Avec

```
FOR A=1 TO 5
```

la partie sera répétée exactement 5 fois, A prenant alors successivement la valeur 1, 2, 3, 4 et 5. Aussi, la variable indiquée après FOR porte le nom de 'compteur'.

Dans ce cas, A est toujours augmenteé de 1 (on dit 'incrémentér de 1'). Cependant, nous pouvons spécifier aussi un pas de progression, et ceci en ajoutant à FOR le terme STEP:

```
10 FOR A=1 TO 6 STEP 2
20 PRINT A; "CECI EST UNE DEMONSTRATION"
30 NEXT A
40 PRINT A
```

Résultat:

```
1 CECI EST UNE DEMONSTRATION
3 CECI EST UNE DEMONSTRATION
5 CECI EST UNE DEMONSTRATION
7
```

Nous voyons comment, à chaque boucle, le compteur A prend successivement les valeurs 1, 3 et 5.

Si A est égal à 7, la limite spécifiée à la ligne 10 (6) étant dépassée, l'ordinateur poursuit le programme par la ligne 40. Si nous laissons imprimer la valeur de A une fois de plus, la valeur 7 est imprimée.

La morale de ce petit programme est évidente: nous devons faire attention quand après l'instruction FOR...NEXT nous utilisons la variable 'Compteur' dans le reste du programme: celle-ci n'a pas nécessairement la valeur finale indiquée dans la ligne contenant FOR!

Encore quelques exemples:

- `FOR A=10 TO 5 STEP-1`
  la série d'instructions est exécutée maintenant pour
  A= 10, 9, 8, 7, 6 et 5

- `FOR A= -15 TO 15 STEP 3`
  la série d'instructions est exécutée pour
  A= -15, -12, -9, -6, -3, 0, 3, 6, 9, 12 et 15

- `FOR A= 1.4 TO 1.7 STEP .05`
  la série d'instructions est exécutée pour
  A= 1.4, 1.45, 1.50, 1.55, 1.60, 1.65 et 1.70

- `FOR A=B TO C STEP K/L`

  Nous remarquons ici que la valeur initiale, la valeur limite ainsi que le pas de progression sont indiqués par des variables; le pas de progression est aussi indiqué par une expression arithmétique (division de 2 variables).

- Des instructions FOR...NEXT peuvent comprendre elles-mêmes d'autres instructions FOR...NEXT. La condition est que la première instruction puisse inclure entièrement l'autre. Ainsi, la construction suivante est admise:

```
FOR K=1 TO 10
...
FOR J=1 TO 5   )
...            }  la boucle intérieure est totale-
...            }  ment imbriquée dans la boucle exté-
...            )  rieure
NEXT J
NEXT K
```

La construction suivante est interdite:

```
FOR K=1 TO 10  )
...            |
FOR J=1 TO 5   |               )
...            } 'BOUCLE K'    |
...            |               |
NEXT K         )               } 'BOUCLE J'
...                            |
NEXT J                         )
```

**ON...GOTO** La dernière instruction de ce chapitre est l'instruction ON...GOTO. Cette instruction fait penser à un interrupteur-sélecteur.

Selon la valeur se trouvant entre ON et GOTO, un certain saut est effectué. Par exemple:

```
10 INPUT K
20 ON K GOTO 30,50
```

```
30 PRINT "K = 1"
40 GOTO 70
50 PRINT "K = 2"
60 GOTO 70
70 END
```

Si K est égal à 1, le saut s'effectuera vers la ligne indiquée par le premier numéro. Si K = 2, le saut s'effectuera vers la ligne indiquée par le deuxième numéro.

Dans cet exemple, seuls 2 numéros de lignes ont été indiqués après GOTO (ligne 20), mais en réalité, il peut y en avoir beaucoup plus. Remarquez que les différentes parties de programme vers lesquelles le saut est effectué, se terminent par des instructions GOTO. L'utilisation d'instructions GOTO est ici inévitable.

# 8 LES TABLEAUX

**Introduction**

Dans certains cas le besoin existe de disposer, dans un programme, d'un grand nombre de variables.

Nous pourrions penser par exemple à un programme de gestion de stocks. Quand nous devons introduire une variable séparée pour chaque article individuel, le programme devient, évidemment, très long.

Nous expliquons ci-dessous comment le MSX-BASIC évite ce problème en offrant la possibilité d'introduire un grand nombre de variables au moyen d'une seule instruction.

**Instruction DIM**

L'instruction par laquelle on introduit facilement un grand nombre de variables, est l'instruction DIM. Celle-ci a toujours la forme suivante:

```
DIM    nom de la variable (nombre)
```

par exemple:

```
DIM A(100)
```

Par cette instruction le MSX-BASIC introduit 101 variables. Le *nom* de chacune de ces variables consiste alors en une lettre et un chiffre entre parenthèses. Ainsi

```
A(0) A(1) A(2) A(3) ... A(99) A(100)
```

constituent exactement les 101 variables appartenant à l'instruction A(100). Dans ce cas nous parlons aussi du tableau A, qui dans cet exemple contient les éléments A(0) à A(100).

Nous pouvons utiliser toutes ces variables de la même façon que nos variables 'ordinaires'.

Exemple:

```
10 DIM A(100)
20 A(3)=6
30 A(27)=5
40 A(98) A(3)+A(27)
50 PRINT A(98)
```

Résultat:

```
11
```

Ce programme, apparemment un peu bizarre, montre que nous avons utilisé les éléments de tableaux A(3), A(27) et A(98), comme s'il s'agissait de variables 'ordinaires' répondant aux noms, par exemple, de A, B etc.

Possibilités des variables de tableaux:

L'élément d'un tableau peut être déterminé également d'une

manière indirecte, en remplaçant son 'index' (nombre entre parenthèses) par une variable ou une expression.
Exemples:

| | |
|---|---|
| A(93) | ici le numéro est indiqué directement par un nombre. |
| A(K) | ici le numéro est indiqué indirectement par une variable. |
| A(K+3) | ici le numéro est indiqué indirectement par une expression. |

Le programme suivant donne une illustration simple:

```
10 DIM B(20)
20 FOR K=1 TO 20
30 B(K)=K
40 NEXT K
50 FOR K=20 TO 1 STEP -1
60 PRINT B(K);
70 NEXT K
```

Résultat:

```
20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
```

A la ligne 10, le tableau B est constitué. Les lignes 20 à 40 indiquent qu'on attribue à B(1) la valeur 1, à B(2) la valeur 2, etc. Les lignes 50 à 70, qui affichent les valeurs des variables B(20), B(19), B(18) etc. prouvent que ces valeurs ont bien été attribuées. Remarquons également comme il est facile d'utiliser l'instruction FOR...NEXT en combinaison avec des tableaux.

Autres remarques au sujet des tableaux:

- Quand un élément de tableau A(6) est utilisée dans un programme qui ne mentionne pas d'instruction DIM, l'ordinateur prend par défaut un tableau de 11 éléments (0 à 10). Dans l'exemple l'ordinateur suppose donc que la variable A(6) appartient au tableau A qui a 10 comme limite supérieure (DIM A(10)).

- Dans un programme, nous pouvons utiliser aussi des tableaux 'pluridimensionnels', c'est-à-dire des tableaux où plusieurs nombres sont indiqués entre parenthèses. Ainsi l'instruction

  ```
  DIM A(3,3)
  ```

  introduit les variables:

  ```
  A(0,0)    A(0,1)    A(0,2)    A(0,3)
  A(1,0)    A(1,1)    A(1,2)    A(1,3)
  A(2,0)    A(2,1)    A(2,2)    A(2,3)
  A(3,0)    A(3,1)    A(3,2)    A(3,3)
  ```

- Nous pouvons introduire plusieurs tableaux en une seule ligne.
  Exemple:

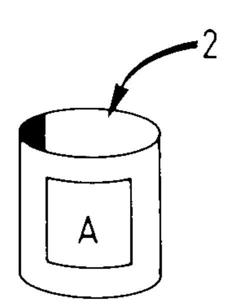
```
10 DIM A(100), P(300), Z(50), P!(30), B%(5)
```

- Nous pouvons éventuellement dégager l'espace réservé en utilisant l'instruction ERASE.
  Exemple:

```
ERASE A,P
```

# 9 CHAÎNES... JOUER AVEC DES LETTRES

**Introduction**



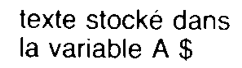

Jusqu'à maintenant, nous avons vu que nous pouvons attribuer un nombre à une variable. Dans l'explication de ce procédé nous avions utilisé une représentation avec une boîte. Ainsi, l'instruction A=2 a été représentée au moyen de l'illustration ci-contre.

Nous pouvons aussi attribuer des séries de lettres à des variables. Une telle série de lettres est appelée une chaîne, et c'est pourquoi ces variables sont appelées des variables chaîne.

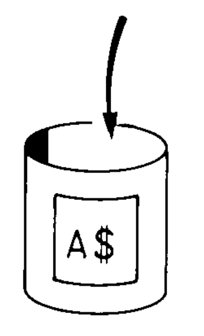
Evidemment, nous devrons indiquer par une méthode spéciale qu'il s'agit d'une variable chaîne. La convention est simple:

quand nous mettons le signe dollar immédiatement après le nom de la variable, il s'agit d'une variable chaîne.

CECI EST UNE CHAINE



texte stocké dans
la variable A $




Ainsi A$, PREMIER$ et TEXTE$ sont des exemples de noms qui concernent des variables chaîne.

Le texte que nous pouvons attribuer à une telle variable chaîne doit être toujours mis entre guillemets.
Le petit programme suivant donne un exemple:

```
10 LET A$= "CECI EST UNE CHAINE"
20 PRINT A$
```

Nous pouvons de nouveau expliquer la ligne 10 par une une figure

Remarquez que, dans la ligne 10, ce texte est placé effectivement entre guillemets. Quand nous regardons le résultat de ce programme, il est évident que ces guillemets ne sont pas imprimés. Voici comment le résultat apparaît:

```
CECI EST UNE CHAINE
```

Nous pouvons donc conclure que les guillemets ne font pas partie de la chaîne elle-même.

Ci-dessous, nous donnons encore quelques exemples de chaînes que nous pouvons attribuer à une variable chaîne.

`"ALLO JEAN"` ceci est une chaîne consistant en neuf caractères, puisque l'espacement entre ALLO et JEAN compte aussi!

"164" — ceci est une chaîne consistant en 3 caractères. Le fait qu'il s'agisse de chiffres est au fond très facheux car l'ordinateur ne voit que des caractères et non des chiffres. Nous verrons plus loin comment nous pouvons convertir dans ce cas particulier ce 'nombre apparent' en un vrai nombre.

"" — ceci est une chaîne ne comportant aucun caractère: il n'y a rien entre les guillemets. Nous verrons plus loin pourquoi une telle 'chaîne vide' peut néanmoins offrir des avantages.

**Enchainer**

Au moyen du signe de l'addition nous pouvons indiquer que deux chaînes doivent être réunies. Les spécialistes parlent alors de concatener.

Exemple:

```
10 A$="MSX-"
20 B$="BASIC"
30 C$=A$+B$
40 PRINT C$
```

Résultat:

```
MSX-BASIC
```

Nous voyons comment, dans les lignes 10 et 20, les chaînes 'MSX-' et 'BASIC' sont attribuées à A$ et B$. Dans la ligne suivante, ces chaînes sont 'additionnées' et attribuées à C$.

**Fonctions de chaînes**

Jusqu'à maintenant, nous n'avons pas pu faire grand-chose avec les chaînes. Au mieux, nous pouvions réunir deux ou plusieurs chaînes pour obtenir ainsi une chaîne un peu plus longue.

Heureusement MSX-BASIC connait un grand nombre de fonctions avec lesquelles toutes sortes d'actions sur des chaînes sont possibles.

Ces fonctions de chaîne se divisent en deux groupes:

**groupe 1:** les fonctions ayant de nouveau une chaine comme résultat. Les noms de ces fonctions finissent toujours par $.

**groupe 2:** les fonctions donnant un nombre comme résultat. La plupart des fonctions de ce groupe utilisent une certaine convention. Cette convention concerne l'attribution de nombres à des lettres et à d'autres caractères. Les noms des fonctions de ce groupe ne finissent pas par $.

Ces deux groupes de fonction sont remarquablement simples. Dans l'aperçu des commandes MSX-BASIC vous en trou-

verez une description détaillée. De plus, chaque fonction est illustrée par un exemple. Nous nous contenterons ici d'énumérer brièvement les fonctions les plus importantes.

LEN détermine le nombre de caractères dans une chaîne.

LEFT$ indique une partie d'une chaîne, dénommée sous-chaîne (partie de gauche).

RIGHT$ indique une sous-chaîne (partie de droite).

MID$ indique une sous-chaîne (en général).

ASC indique la valeur ASCII du premier caractère.

CHR$ indique le caractère correspondant à la valeur ASCII donnée.

VAL convertit une chaîne numérique en un nombre correspondant.

STR$ convertit un nombre en une chaîne correspondante.

SPACE$ permet d'imprimer un nombre donné d'espaces.

INSTR permet de chercher une sous-chaîne dans une chaîne donnée.

**INKEY$ et un petit jeu**

INKEY$ n'est pas une fonction, mais une variable. Si l'ordinateur rencontre le terme INKEY$ quand il exécute un programme, il 'regarde' le clavier. Quand une touche est enfoncée, le caractère concerné est attribué à INKEY$. Quand nous n'enfonçons pas de touches, la chaîne vide "" sera attribuée à INKEY$.

Le programme suivant explique comment INKEY$ est utilisée pour un jeu de réflexe simple:

```
10 PRINT "POUSSEZ UNE TOUCHE"
20 PRINT "QUAND L'ECRAN"
30 PRINT "MONTRE LA VALEUR 25"
40 FOR K=1 TO 50
50 PRINT K
60 A$=INKEY$
70 IF A$ <> "" THEN GOTO 90
80 NEXT K
90 PRINT "FIN"
```

**Encore quelques remarques**

Nous venons d'étudier les fonctions principales des chaînes. Une description de toutes ces fonctions est donné dans l'aperçu des commandes MSX-BASIC.
Faisons encore les remarques suivantes:

- Comme avec les nombres, nous pouvons introduire des tableaux de chaîne, par exemple:

```
DIM A$(100)
```

- Quand il a été mis en marche, l'ordinateur a réservé une certaine capacité de mémoire pour le stockage de chaînes. Normalement, 200 caractères peuvent être stockés ainsi. Nous pouvons étendre cette capacité au moyen d'une instruction CLEAR, par exemple:

```
CLEAR 500
```

Avec cette instruction, nous réservons un espace de 500 caractères.

- La chaîne que nous pouvons attribuer à une expression chaîne a une longueur maximum de 255 caractères.

# 10 SOUS-PROGRAMMES GOSUB...RETURN ET DEF FN

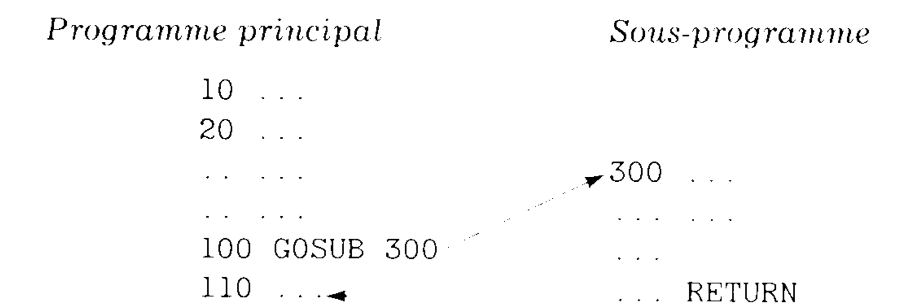
**GOSUB** Un sous-programme est une partie de programme qui peut être appelée à partir d'un endroit quelconque du programme. Le saut vers le sous-programme s'effectue toujours au moyen de l'instruction:

```
GOSUB numéro de ligne
```

Lorsque l'ordinateur rencontre cette instruction, il poursuit le programme à la ligne indiquée. Après avoir rencontré le mot:

```
RETURN
```

l'ordinateur retourne au programme initial. Le schéma qui suit montre clairement comment cela se passe.




Nous voyons comment le sous-programme est appelé à la ligne 100. Il commence à la ligne 300.

Lorsque l'ordinateur rencontre RETURN, il retourne à la ligne 110. Autrement dit, l'ordinateur poursuit le programme initial. Nous avons expliqué toute la procédure, mais il convient d'ajouter qu'un sous-programme peut également contenir un saut vers un autre sous-programme.

Le programme suivant va nous le montrer. Ce programme nous permet de jouer au jeu des '13 allumettes'. Chaque joueur peut prélever à tour de rôle 1, 2 ou 3 allumettes dans le tas d'allumettes. Celui qui est obligé de prendre la dernière allumette a perdu.

```
 10 PRINT "A VOUS DE COMMENCER"
 20 L=13
 30 GOSUB 80
 40 GOSUB 80
 50 GOSUB 80
 60 PRINT "AH AH, J'AI GAGNE"
 70 END
 80 REM DEBUT DU SOUS-PROGRAMME
 90 INPUT K: IF K<0 OR K>3 THEN GOTO 90 ELSE A=4-K
100 PRINT "J'EN PRENDS...";A
110 L=L-K-A
120 PRINT "NOMBRE D'ALLUMETTES RESTANT=";L
130 PRINT "A VOTRE TOUR"
140 RETURN
```

Vous avez certainement noté que le sous-programme est séparé du programme principal au moyen de END. Nous évitons de cette manière que le sous-programme ne soit exécuté sans l'instruction GOSUB. Pour être clair : les sous-programmes ne peuvent être exécutés qu'au moyen de l'instruction GOSUB.

**DEF FN** L'instruction DEF FN nous permet de définir une fonction nous-mêmes.
L'instruction DEF FN a toujours la forme suivante:

DEF FN nom (série de variables séparées par des virgules)=expression dans laquelle ces variables figurent

Exemple:

```
DEF FNA (X,Y)=X^2+Y^2
```

le nom de la fonction consiste en une seule lettre (dans notre exemple la lettre A). Dans cet exemple, la fonction A est définie par:

$X^2+Y^2$

Cette fonction est alors appelée par l'expression FNA, ou par le terme FN suivi du nom de la fonction. Le programme suivant le montre:

```
10 DEF FNA(X,Y)=X^2+Y^2
20 A=3
30 B=4
40 C=FNA(A,B)
50 PRINT C
```

Résultat:

```
25
```

# EXTENSIONS DE MSX2-BASIC

## 1 SON: BEEP, PLAY ET SOUND

**BEEP** BEEP veut dire 'un court son sifflant'.

Le programme suivant illustre cette instruction:

```
10 PRINT "COMMENCEZ"
20 FOR K=1 TO 1000
30 PRINT K
40 NEXT K
50 BEEP
60 PRINT "TERMINE"
```

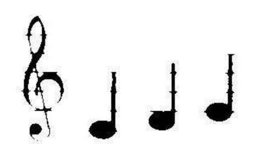
**PLAY** L'instruction PLAY sert à jouer une mélodie.

C'est une instruction extrêmement puissante qui nous permet entre autres d'indiquer:

- le rythme
- l'octave
- la durée d'une note
- les silences
- le volume
- certains effets sonores

PLAY peut être utilisée comme instruction, mais aussi comme commande.
Introduisez seulement:

```
PLAY "CDE"
```



Quand la touche RETURN est appuyee, nous entendons

brièvement, les notes do - ré - mi et plus précisément celles qui appartiennent à l'octave avec la clef de sol. Dorénavant nous indiquerons cette octave par 'octave 4'.

Le programme suivant utilise l'instruction PLAY pour faire de notre ordinateur MSX un petit orgue.

```
10 A$=INPUT$(1)
20 IF INSTR("ABCDEF",A$)>0 THEN PLAY A$
30 GOTO 10
```

En ligne 10, la touche appuyée est attribuée à A$; celle-ci est alors utilisée dans l'instruction PLAY, si elle correspond à une note, etc...

Notre petit orgue est encore loin d'être parfait. Entre autres, nous ne pouvons pas du tout régler le rythme. Heureusement, il y a de nombreuses possibilités pour arranger cela.

**Plusieurs générateurs sonores**

Quand nous insérons, après PLAY, trois chaînes séparées par des virgules, l'ordinateur MSX utilise un générateur sonore séparé pour chaque chaîne. par exemple, avec

```
PLAY "CDE"
```

un générateur sera utilisée, et avec

```
PLAY "CDE", "EFG"
```

deux générateur seront utilisés simultanément, et avec

```
PLAY "CDE", "EFG", "GAC"
```

les trois générateurs seront utilisés simultanément. Cela implique que les notes | do | mi sol et ensuite | ré | fa | la et finalement | ré | sol | do seront produites simultanément.

**Tempo**

Le tempo est indiqué par un code T.
Donnez la commande suivante:

```
PLAY "T200CDE"
```

Les notes | do | ré | mi sont jouées maintenant à un tempo plus élevé qu'avant. C'est correct, parce que, par défaut, l'ordinateur prend la 'position' T120 et T200, ce qui donne donc un tempo plus élevé. La valeur T peut se situer entre 32 et 256.

**Volume**

Le volume est indiqué par un code V.
Exemple:

```
PLAY "T200V13CDE"
```

Nous entendons la même mélodie que dans l'exemple précédent, seulement le bouton du volume V est apparemment plus fort. C'est correct aussi, puisque par défaut l'ordinateur prend, la 'position V8' et V13 qui veut dire un plus grand volume. Le code V peut varier entre 0 (minimum) et 15 (maximum).

**Les notes et l'octave**

Les notes sont indiquées par les lettres C, D, E, F, G, A et B. En cas de demi-tons, par exemple, nous utilisons le signe # (ou +). Exemple:

```
PLAY "CC#" et aussi PLAY "CC+"
```

font entendre successivement le do et le do dièse. La question est maintenant 'comment indiquer un do, plus haut?' Le do plus haut appartient à l'octave suivante. Nous avons déjà dit que l'ordinateur prenait par défaut l'octave 4 (O4).

Pour produire le do, nous devons donc indiquer O5.
Exemple:

```
PLAY "CDEO5CDE"
```

fait entendre deux fois la série | do | ré | mi, mais la deuxième série est jouée sur une octave supérieure.

Le code d'octave (code O) peut varier entre 1 (minimum) et 8 (maximum): notre ordinateur MSX peut donc jouer sur 8 octaves.

**Durée des notes**

Jusqu'ici, toutes les notes avaient la même durée. Cependant, quand nous ouvrons un livre de musique, nous voyons que les notes peuvent avoir des durées différentes.

En musique, nous indiquons cela en remplissant ou non la note et également en dessinant des queues (appelées 'croches') sur les bâtons.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *signe* | | | | | | | |
| *durée en secondes* | 4 | 2 | 1 | [illegible] | [illegible] | [illegible] | [illegible] |
| *code L* | 1 | 2 | 4 | 8 | 16 | 32 | 64 |

Nous indiquons la durée d'une note par le code L. L'ordinateur prend par défaut L4, et, cela revient alors à une double croche.

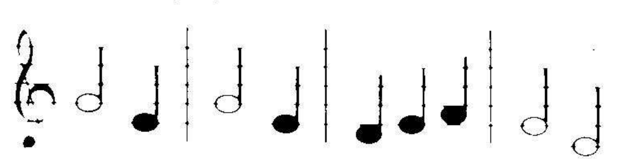
Ci-dessous, nous expliquons le code L au moyen d'une mélodie bien connue.

Converti en une commande PLAY cela devient:

```
PLAY "L2GL4EL2GL4EDEFL2EL4C"
```

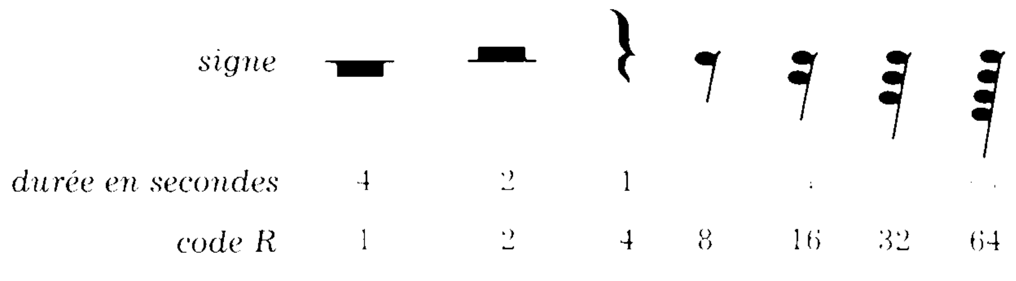
Nous pouvons également écrire cela d'une manière plus courte, à savoir en indiquant la durée immédiatement après une note, donc pour notre exemple:

```
PLAY "G2EG2EDEFE2C"
```



**Silences** En musique, nous rencontrons également des signes de repos. Ceux-ci indiquent qu'aucune note ne doit être jouée pendant une certaine période, ou mieux, pendant un certain nombre de secondes. Nous utilisons à cette fin le code R.

L'aperçu suivant montre les signes de musique utilisés à cette fin en combinaison avec les codes R.




Exemple:

```
10 PLAY "CDER4DECDECR2": GOTO 10
```

Après le premier sol, il y a un silence d'une seconde et après le dernier do il y un silence de deux secondes. En utilisant PLAY de cette façon dans un programme, nous pouvons encore mieux respecter le tempo de la musique.

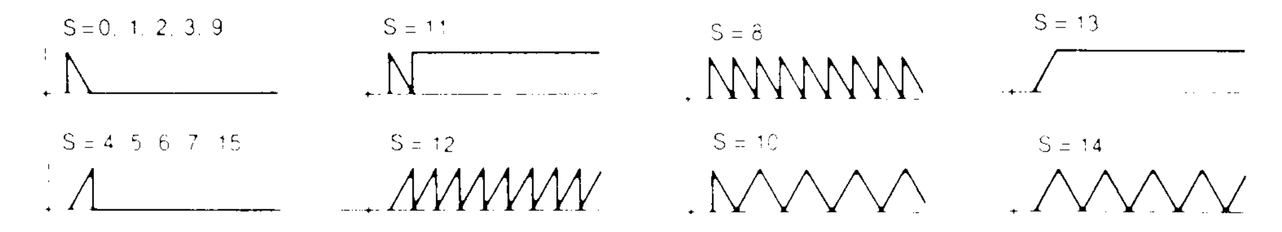
Nous interrompons cette belle musique par CTRL/STOP. Dans ce paragraphe sur les silences nous remarquons enfin que nous pouvons mettre un point derrière une note. La durée de cette note sera lors une fois et demie plus longue, par exemple:

```
PLAY "CD.E"
```

De cette façon aussi, nous pouvons déterminer le rythme de notre musique.

**Les codes S et M** Les derniers codes que nous pouvons utiliser avec l'instruction PLAY sont les codes S et M. Le code S concerne le timbre que nous voulons donner et le code M détermine dans quelle mesure le dessin de ce timbre doit être répété.

Le code S indique comment notre son doit être amplifié et affaibli selon un certain dessin. A cette fin nous utilisons les images suivantes:




Le code M indique alors la longueur du cycle dans lequel la forme choisie avec S doit être répetée. L'ordinateur donne par défaut la valeur 1 à S et la valeur 255 à M (M peut varier de 1 à 65535).

Exemples:

```
PLAY "S10M510CDE"
```

et

```
PLAY "S8M6000CDE"
```

Dans le premier exemple, nous entendons à chaque note un son heurtant. Dans le deuxième exemple, c'est comme si les notes étaient jouées sur un piano. Essayez maintenant vous-même les différentes combinaisons pour découvrir les différentes possibilités des codes S et M.

**Un exemple élaboré**

L'exemple suivant laisse jouer par l'ordinateur le canon 'Frère Jacques'. Nous pouvons voir comment les chaînes utilisées dans l'instruction PLAY, sont construites, à l'avance, au moyen des expressions chaîne.

```
 10 CLEAR 500
 20 A$="L4CDECR64CDEC"
 30 B$="EFG2EFG2"
 40 C$="L8GAGFL4ECL8GAGFL4EC"
 50 D$="CO3GO4C2C4O3GO4C2"
 60 W$=A$+B$+C$+D$
 70 W1$=W$
 80 W2$="R1R1"+W$
 90 W3$="R1R1"+W2$
100 PLAY W1$,W2$,W3$
```

(Remarque: Dans la chaîne de la ligne 50, des O majuscules sont utilisés)

L'instruction CLEAR est utilisée pour créer suffisamment d'espace pour le stockage de chaînes. Dans les lignes 20, 30, 40 et 50, la mélodie est indiquée dans les morceaux caractéristiques. La ligne 60 donne la combinaison des chaînes et, ainsi, de la mélodie entière. Les lignes 80 et 90 déterminent les deuxième et troisième voix.

**SOUND**

La dernière instruction que nous verrons dans ce chapitre est l'instruction SOUND. Nous ne pouvons comprendre cette instruction que si nous savons que la puce est équipée d'un certain nombre de registres (voir chapitre 1). En mettant certaines valeurs dans ces registres, cette puce produit toutes sortes de sons.

Notre puce à son est munie de 13 registres qui ont la signification suivante

| *Registre numéro* | *utilisation* |
|---|---|
| **0,1** | la combinaison du contenu des registres 0 et 1 détermine la fréquence du générateur de fréquence. |
| **2,3** | idem, mais pour le générateur B. |
| **4,5** | idem, mais pour le générateur C. |
| **6** | détermine la fréquence du son à produire. La valeur peut varier entre 0 et 31. |
| **7** | détermine le bruit que donne le générateur. |
| **8** | détermine le volume du canal A (valeur entre 0 et 15) |
| **9** | comme 8, mais pour générateur canal B. |
| **10** | comme 8, mais pour générateur C. |
| **11,12** | détermine le timbre, à savoir la modulation d'une note. |
| **13** | détermine l'enveloppe (crescendo et decrescendo). |

Pour calculer la fréquence d'un son, nous utilisons le programme auxiliaire suivant:

```
10 INPUT "FREQUENCE DESIREE": F
20 A=1789772.5
30 B=INT(A/(16*F))
40 C=INT(B/256)
50 D=B-C
60 PRINT D,C
```

Les valeurs de D et C sont alors les valeurs qui doivent être placées dans les premier et deuxième registres pour obtenir la note voulue.

Supposons que nous voulions produire un 'la' (fréquence 440). Notre programme produit les valeurs suivantes:

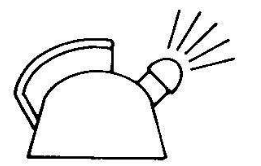
```
254      0
```

Ceci mène alors aux instructions SOUND suivantes:

```
10 SOUND 0,254
20 SOUND 1,0
30 SOUND 8,11
```

Nous entendons maintenant le la (essayez aussi PLAY 'A'). Nous pouvons interrompre la note par CTRL/STOP.

**Souffle** En pratique, nous utiliserons l'instruction SOUND pour créer toutes sortes d'effets sonores, comme des sirènes perçantes et des bruits sourds. Les exemples suivants donnent quelques possibilités:



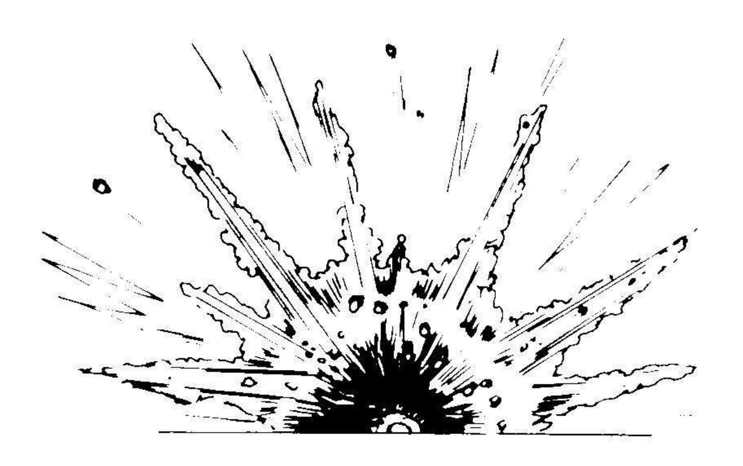
```
10 REM BOUILLOIRE A SIFFLET
20 FOR K=255 TO 0 STEP-1
30 PRINT K
40 SOUND 0,K
50 SOUND 1,0
60 SOUND 8,10
70 NEXT K
```

En omettant la ligne 30, nous obtenons une variation accélérée du son.

Avec l'exemple suivant, nous pouvons essayer encore plusieurs effets:

```
10 INPUT K,L,M
20 SOUND 0,250:SOUND 1,0
30 SOUND 6,K:SOUND 7,L:SOUND 13,M
40 FOR J=15 TO 0 STEP -0.05
50 SOUND 8,J
60 NEXT J
```

Pour K=10, L=10 et M=10 cela donne un decrescendo, mais pour K=20, L=20 et M=20 nous entendons une explosion.



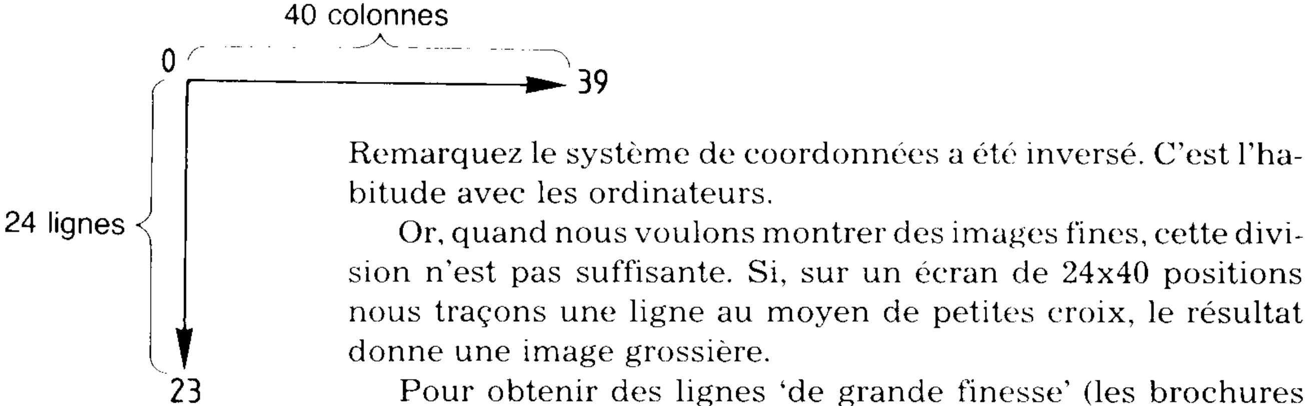
# 2 REPRÉSENTATIONS GRAPHIQUES: SCREEN, PSET, COLOR ET PRESET

**Introduction** Une visualisation n'est qu'un mot savant pour une image. Nous créons des images au moyen de points, de lignes et de courbes.

L'ordinateur MSX possède un très grand nombre de possibilités pour montrer toutes sortes de belles images sur l'écran. Ceci est important, parce que beaucoup d'applications dépendent précisément de l'emploi d'images captivantes.

**SCREEN** Pour comprendre comment créer des images avec un ordinateur MSX, il faut d'abord parler de la division de l'écran (en anglais: screen).

Pour toutes les images montrées jusqu'ici, l'ordinateur utilisait une division en 24 lignes, dont chacune pouvait comporter 40 caractères (après le commande WIDTH 40).



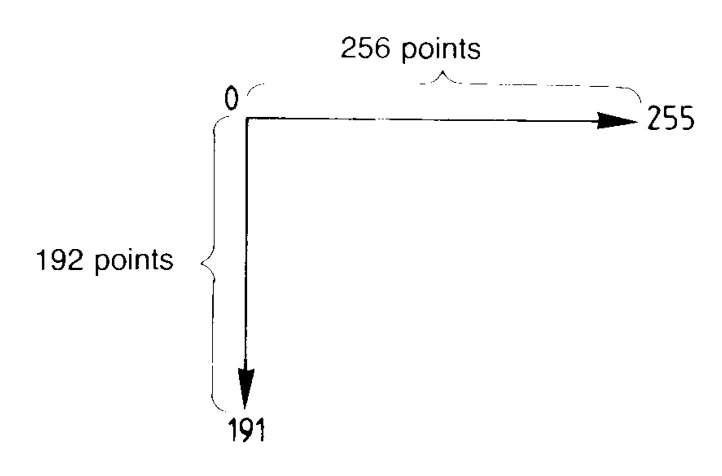

Remarquez le système de coordonnées a été inversé. C'est l'habitude avec les ordinateurs.

Or, quand nous voulons montrer des images fines, cette division n'est pas suffisante. Si, sur un écran de 24x40 positions nous traçons une ligne au moyen de petites croix, le résultat donne une image grossière.

Pour obtenir des lignes 'de grande finesse' (les brochures parlent d'une 'haute résolution'), nous devons commander à l'ordinateur une autre division d'écran.
Après l'instruction:

```
SCREEN 2
```

l'ordinateur utilise la division suivante:



Dorénavant, nous appellerons chaque carré un point d'image (ou pixels) et les coordonnées de chaque point seront indiquées par une valeur X et une valeur Y selon la convention suivante : l'axe horizontal est l'axe des X et l'axe vertical, l'axe des Y.

X peut varier de 0 à 255 et Y de 0 à 191.

Le point (0,0) est donc le point en haut à gauche et le point (255,191) est le point en bas à droite.

Quand l'ordinateur a été mis, au moyen de l'instruction SCREEN 2, dans la position 'grille fine' nous pouvons dessiner des points et des lignes en utilisant les instructions prévues à cette effet.

**PSET** Nous commençons par l'instruction PSET (pointset, c'est-à-dire mettez un point), avec laquelle nous pouvons mettre des points sur l'écran.
Dans sa forme la plus simple, cette instruction est la suivante:

PSET (coordonnée x, coordonnée y)

Commençons immédiatement par un exemple:

```
10 INPUT "X=";X
20 INPUT "Y=";Y
30 SCREEN 2
40 PSET (X,Y)
50 GOTO 50
```

Les lignes 10 et 20 attribuent une valeur à X et à Y. La ligne 30 comporte l'instruction nécessaire SCREEN 2. Ensuite, le point est placé au moyen de l'instruction PSET. La dernière instruction fait boucler le programme car sans elle nous ne verrions rien. (a la fin du programme, MSX-BASIC re-initialize l'écran en SCREEN 0).

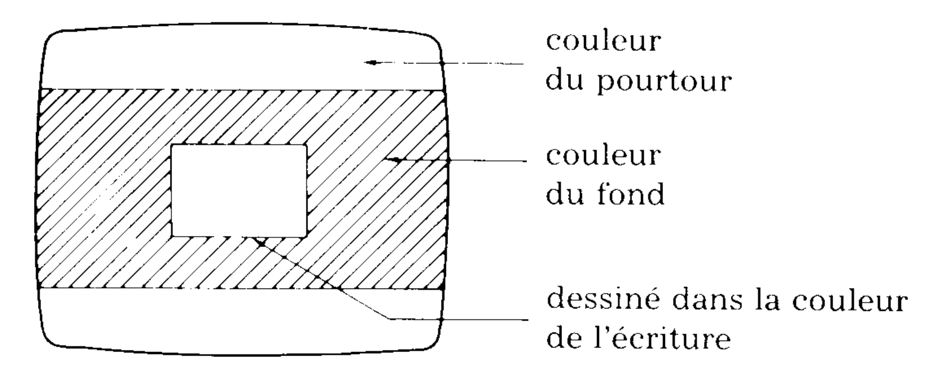
Ainsi, nous ne pouvons interrompre le programme qu'en appuyant sur les touches STOP et CTRL (CTRL/STOP).

Votre ordinateur possède plusiers modes SCREEN qui permettent d'adapter l'écran au mieux desimages à reproduire. Les différentes possibilités figurent dans le tableau suivant.

| *Mode-SCREEN* | *Description* | *Couleur* |
|---|---|---|
| 0 | Pour l'affichage de texte : 40 × 24. | 2 parmi 512 |
| | Avec la commande WIDTH 80 : 80 × 24. | - - - |
| 1 | Pour l'affichage de texte : 32 × 24. | - - - |
| 2 | Graphique : 256 × 192 pixels (points). | 16 parmi 512 |
| 3 | Graphique : 64 × 48 blocs. | - - - |
| 4 | Graphique : 256 × 192 pixels. | - - - |
| 5 | Graphique : 256 × 212 pixels. | - - - |
| 6 | Graphique : 512 × 212 pixels. | 4 parmi 512 |
| 7 | Graphique : 512 × 212 pixels. | 16 parmi 512 |
| 8 | Graphique : 256 × 212 pixels. | 256 |

**COLOR** Avant d'expliquer comment utiliser les couleurs (colors en américain), nous allons commencer par définir les expressions 'écriture', 'fond' et 'pourtour'.

L'illustration suivante montre comment l'écran est divisé.

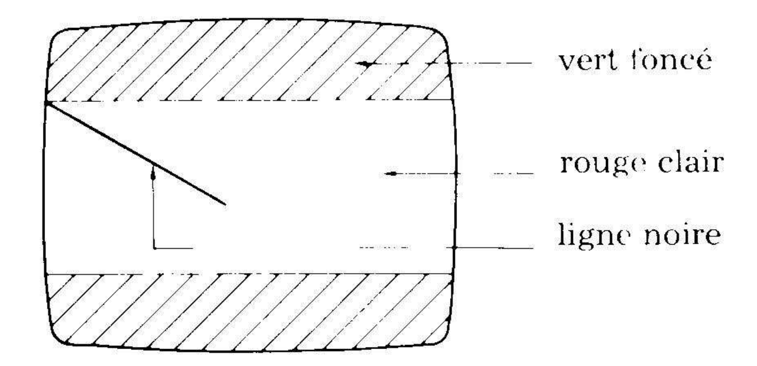



Le mot 'écriture' signifie les signes, figures, symboles, etc. que nous pouvons reproduire sur l'écran. Ces éléments peuvent s'afficher sur l'écran dans une couleur déterminée. C'est pourquoi nous parlons de la 'couleur de l'écriture'.

La couleur sur laquelle nous reproduisons ces signes, figures, symboles, etc. s'appelle la couleur du fond. Enfin, nous constatons qu'il existe également un pourtour, dont nous pouvons changer la couleur (excepté SCREEN 0).
Un exemple

```
10 SCREEN 2
20 COLOR 1,9,12
30 CLS
40 FOR K=1 TO 100
50 PSET (K,K)
60 NEXT
70 GOTO 70
```

La représentation suivante apparaît :




Les instructions des lignes 40 à 60 entraînent l'affichage de 100 points, qui forment la ligne noire.

Cet exemple comporte une instruction COLOR. Cette instruction a la forme suivante :

`COLOR` écriture, fond, pourtour

Le tableau suivant nous permet d'obtenir les couleurs standard (sauf en mode SCREEN 8):

| *chiffre/ couleur* | *chiffre/ couleur* | *chiffre/ couleur* | *chiffre/ couleur* |
|---|---|---|---|
| 0 transparent | 4 bleu foncé | 8 rouge | 12 vert foncé |
| 1 noir | 5 bleu clair | 9 rouge clair | 13 magenta |
| 2 vert | 6 rouge foncé | 10 jaune foncé | 14 gris |
| 3 vert clair | 7 bleu d'azur | 11 jaune clair | 15 blanc |

Nous constatons que dans notre exemple la couleur de l'écriture est effectivement le noir (1), que le fond est rouge clair (9) et que le pourtour est vert foncé (12).

Voici d'autres exemples :

`COLOR 1` : écrire (1), autrement dit: dessinez des lignes noires.

`COLOR ,2` : changez la couleur du fond en vert. Remarquez que nous mettons une virgule devant le nombre.

`COLOR 1,2,11` : écrire en noir (1), donner la couleur verte (2) au fond et la couleur jaune claire (11) au pourtour.

COLOR ,,11 : changez seulement la couleur du pourtour. Remarquez que le nombre est précédé de deux virgules.

Signalons enfin que pour modifier la couleur du fond, nous devons toujours mettre une instruction CLS (CLEAR SCREEN : effacez l'écran) après l'instruction COLOR.

**COLOR**

De nombreux peintres se servent d'une palette pour mélanger des couleurs et obtenir la couleur désirée. En fait, le peintre mélange un grand nombre de 'points' de couleur. La théorie des couleurs nous apprend qu'il suffit de mélanger trois couleurs de base pour obtenir toutes les autres couleurs. C'est ainsi qu'en télévision couleur les différents nuances de couleur sont obtenues en modulant l'intensite des trois faisceaux de lumière : un rouge, un vert et un bleu.

Les couleurs du tableau précédent ont donc été obtenues en effectuant les mélanges suivants :

| *Code couleur* | *Couleur* | | *Intensité* | |
|---|---|---|---|---|
| | | *Rouge* | *Vert* | *Bleu* |
| 0 | Transparent | 0 | 0 | 0 |
| 1 | Noir | 0 | 0 | 0 |
| 2 | Vert | 1 | 6 | 1 |
| 3 | Vert clair | 3 | 7 | 3 |
| 4 | Bleu foncé | 1 | 1 | 7 |
| 5 | Bleu clair | 2 | 3 | 7 |
| 6 | Rouge foncé | 5 | 1 | 1 |
| 7 | Cyan | 2 | 6 | 7 |
| 8 | Rouge | 7 | 1 | 1 |
| 9 | Rouge clair | 7 | 3 | 3 |
| 10 | Jaune foncé | 6 | 6 | 1 |
| 11 | Jaune clair | 6 | 6 | 4 |
| 12 | Vert foncé | 1 | 4 | 1 |
| 13 | Magenta | 6 | 2 | 5 |
| 14 | Gris | 5 | 5 | 5 |
| 15 | Blanc | 7 | 7 | 7 |

Nous voyons ainsi que le rouge clair dont le code couleur est 9 s'obtient en mélangeant du rouge, du vert et du bleu selon les intensités 7, 3 et 3. A défaut d'instruction spéciale de notre part, le code couleur 9 correspondra toujours à ce rapport.

Toutefois, il est possible de définir soi-même un code couleur en utilisant l'instruction COLOR=. Cette instruction a la forme suivante :

COLOR= (code couleur, intensité rouge, intensité vert, intensité bleu). Pour illustrer ce qui précède, ajoutons l'instruction suivante au programme qui nous a permis de dessiner la ligne noire sur fond rouge clair :

```
15 COLOR=(9,1,2,7)
```

Le programme complet se présente comme suit :

```
10 SCREEN 2
15 COLOR=(9,1,2,7)
20 COLOR 1,9,12
30 CLS
40 FOR K=1 TO 100
50 PSET (K,K)
60 NEXT K
70 GOTO 70
```

Nous avons donc défini nous-mêmes le code couleur 9 en choisissant les intensités 1, 2 et 7. Il s'agit vraiment d'une nouvelle couleur, qui ne figure pas encore dans notre tableau des couleurs standard !

Après avoir donné la commande RUN, nous constatons que notre fond rouge clair s'est transformé en bleu vif.

Les autres couleurs, c'est-à-dire la couleur de l'écriture (la ligne noire) et celle du pourtour (vert) n'ont pas changé. C'est normal, puisque les codes couleur 1 et 12 n'ont pas été redéfinis. Eventuellement, nous pouvons attribuer à chacun de ces codes la couleur de notre choix.

Si nous voulons retrouver les couleurs initiales (celles du tableau), nous devons utiliser l'instruction COLOR=NEW.
En ajoutant au programme précédent la ligne :

```
27 COLOR-NEW
```

nous retrouvons la palette de couleurs initiales.

**Indication de couleur après une instruction graphique**

L'instruction PSET est une instruction graphique parce qu'elle nous permet de créer des images. Dans le chapitre suivant, nous traiterons également des instructions LINE et CIRCLE. Il s'agit également d'instructions graphiques. Nous pouvons introduire directement des indications de couleur dans une instruction graphique de ce type :

- en plaçant un code couleur immédiatement après l'instruction. On définit ainsi une couleur d'écriture.
- en ajoutant une opération logique (AND et OR sont des exemples d'opérations logiques.) Vous trouverez la liste complète des opérations logiques possibles dans l'aperçu des commandes MSX-BASIC.

Prenons un exemple:

```
10 SCREEN 5
20 COLOR 15,4,4
30 CLS
40 FOR K=1 TO 100
50 PSET (K,K),1
60 NEXT K
70 GOTO 70
```

Nous voyons apparaître une ligne noire, puisque le code couleur 1 correspond au noir.

Nous constatons qu'en plaçant un code couleur dans l'instruction PSET, la couleur de l'écriture indiquée dans l'instruction COLOR (l'écriture avait 15 pour code) est modifiée.

**Opération logique**

A présent, introduisons également une opération logique. A titre d'exemple, transformez la ligne 50 comme suit :

```
50 PSET (K,K),1,OR
```

Nous constatons la présence de l'opération logique OR, placée après l'indication de couleur.

Pour déterminer à présent la couleur que nous allons obtenir, nous devons d'abord expliquer deux concepts : les concepts SCREEN COLOR (SC) et DESTINATION COLOR (DS). SC est toujours la couleur de l'écriture (déterminée par PSET) avec laquelle on travaille et DS la couleur du fond, telle qu'elle apparaît sur l'écran.

Dans notre exemple, SC est égal à 1 et DS est égal à 4. Le terme OR entraîne la valeur 1 or 4 = 5, c'est-à-dire bleu clair (Remarque: 5 OR 3 = 7 et non pas 8).

# 3 REPRÉSENTATIONS GRAPHIQUES: LINE, DRAW, CIRCLE ET PAINT ET COPY

**Introduction**

Dans ce chapitre, abordons d'autres instructions graphiques.

Les instructions LINE et DRAW nous offrent la possibilité de tracer de manière simple des lignes droites, tandis que CIRCLE nous permet de dessiner des arcs (parties d'un cercle ou d'une ellipse) sur l'écran.

Toutes ces instructions ne fonctionne qu'en modes graphiques tel que nous l'avons vu au chapitre précédent.

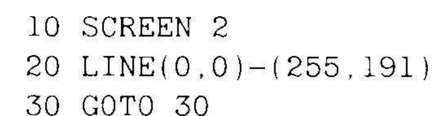
**LINE**

La syntaxe la plus simple de l'instruction LINE est:

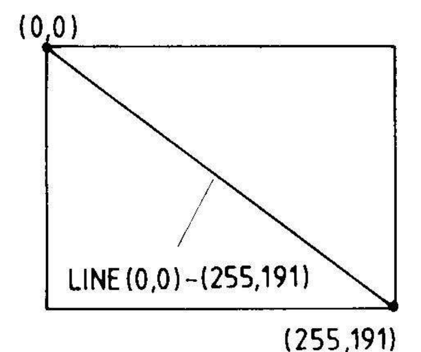
```
LINE (X1,Y1)-(X2,Y2)
```

(X1,Y1) indique le point de départ de la ligne et (X2,Y2) le point d'arrivée.

Exemple:

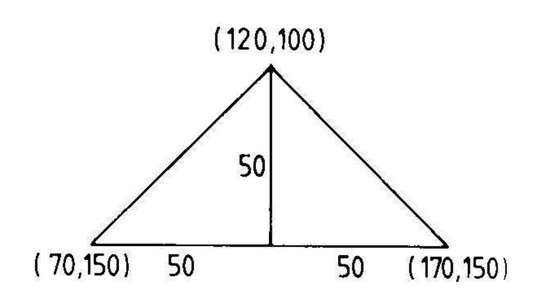


```
10 SCREEN 2
20 LINE(0,0)-(255,191)
30 GOTO 30
```




Comme résultat, nous voyons une ligne diagonale sur l'écran.

Dans l'exemple suivant, nous dessinons un triangle, et ceci selon la définition suivante:




Programme:

```
10 SCREEN 2
20 LINE ( 70,150)-(170,150)
30 LINE (170,150)-(120,100)
40 LINE (120,100)-( 70,150)
50 GOTO 50
```

Si la position de départ est omise, la ligne commencera à partir du dernier point atteint. Le programme devient alors:

```
10 SCREEN 2
20 LINE ( 70,150)-(170,150)
30 LINE -(120,100)
40 LINE -( 70,150)
50 GOTO 50
```

En outre, nous pouvons donner une couleur à une ligne, et ceci selon la même méthode qu'avec PSET, c'est-à-dire en mettant une virgule et un code couleur:

LINE (X1,Y1)-(X2,Y2), code couleur

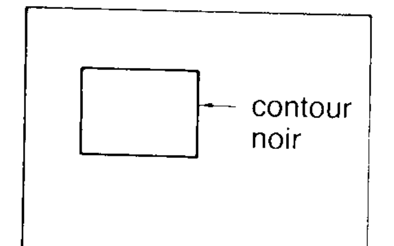
L'instruction LINE nous permet de dessiner facilement des boîtes. A cette fin nous ajoutons ,B à l'instruction, donc:

LINE (X1,Y1)-(X2,Y2), couleur, B

Exemple:

```
10 SCREEN 2
20 LINE (50,50)-(100,100),1,B
30 GOTO 30
```

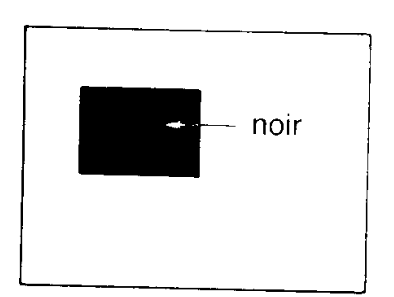
donne comme résultat:




Remarquez que nous avons défini la boîte en spécifiant dans l'instruction LINE deux points angulaires diagonalement opposés. Si, au lieu de B, nous mettons BF, le carré entier est peint, par exemple:

```
10 SCREEN 2
20 LINE (50,50)-(100,100),1,BF
30 GOTO 30
```

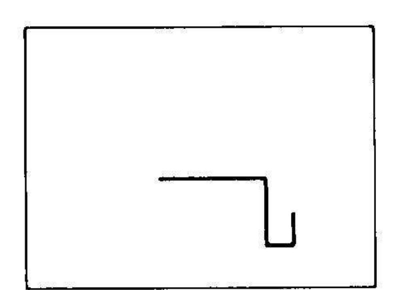
donne:




Enfin, après BF, nous pouvons aussi placer une opération logique comme AND ou OR. De cette manière, on peut influencer la couleur. On trouvera un exemple de l'effet d'une telle opération logique là où l'instruction PSET est traitée. Dans l'aperçu des commandes MSX-BASIC, on trouvera dans l'explication consacré à l'instruction PSET une liste complète des diverses possibilités que permettent les expressions logiques.

**DRAW** L'instruction DRAW est une instruction très puissante pour dessiner des lignes horizontales, verticales et inclinées.

Donnons un exemple simple:



```
10 SCREEN 2
20 PSET (127,95)
30 DRAW "R50D50R25U25"
40 GOTO 40
```

Résultat:

La ligne 20 donne le point de départ de notre ligne. L'instruction suivante, DRAW, indique, au moyen d'une chaîne, ce qui doit être dessiné, à savoir:

*R50* tracez une ligne en déplaçant 'le stylo' de 50 points vers la droite (R vient de Right).

*D50* maintenant, descendez de 50 points (D vient de 'down', ce qui veut dire 'en bas')

*R25* ensuite, déplacez de nouveau de 25 points vers la droite.

*U25* puis 25 points vers le haut (U vient de 'up', ce qui veut dire 'en haut').

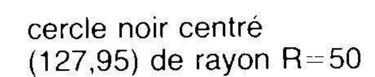
L'aperçu des commandes MSX-BASIC donne une liste complète des possibilités de DRAW.

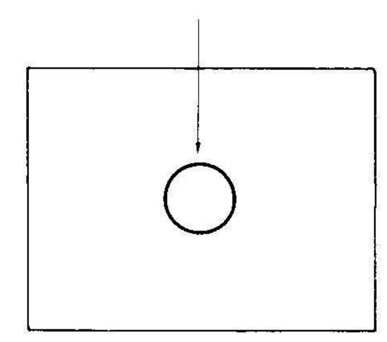
**CIRCLE** C'est la dernière instruction que nous verrons pour dessiner des lignes, dans ce cas des courbes. La forme générale est:

`CIRCLE (X,Y) ,` rayon, couleur, angle de départ, angle d'arrivée, coefficient d'aplatissement

*avec:*

| | |
|---|---|
| (X,Y) | coordonnées du centre |
| rayon | valeur du rayon |
| couleur | code de couleur |
| angle de départ | angle à partir duquel l'arc doit être dessiné (indiqué en radians: $0..2\pi$) |
| angle d'arrivée | angle jusqu'ou l'arc doit être dessiné (indiqué en radians: $0..2\pi$) |
| coefficient d'aplatissement | nombre indiquant dans quelle mesure le cercle est aplati, dans les modes SCREEN 2, 3 et 4. En donnant la valeur 1.36, on obtient un cercle dans les modes SCREEN 5 et 8. En utilisant la valeur 0.68, on obtient un cercle dans les modes SCREEN 6 et 7. |



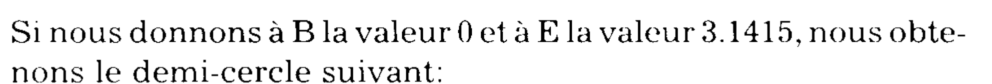
Donnons quelques exemples:

```
10 SCREEN 2
20 CIRCLE (127,95),50,1,,,1.4
30 GOTO 30
```

Résultat:

L'effet des angles de départ et d'arrivée peut être montré au moyen du programme suivant:

```
10 INPUT B
20 INPUT E
30 SCREEN 2
40 CIRCLE (127,95),50,1,B,E,1.4
50 GOTO 50
```

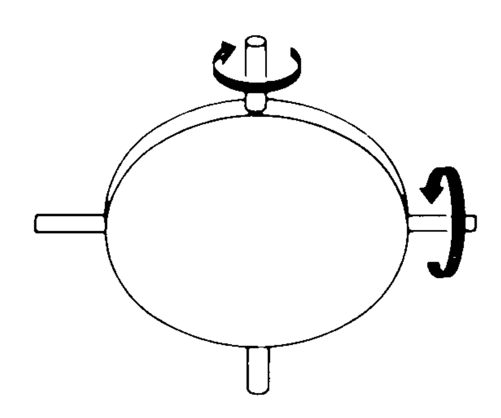


Si nous donnons à B la valeur 0 et à E la valeur 3.1415, nous obtenons le demi-cercle suivant:

demi-cercle noir

point de départ

Le dernier nombre de l'instruction CIRCLE détermine le coefficient d'aplatissement. Considérons l'image suivante:



Quand le disque tourne autour d'un des axes, nous voyons le disque sous un autre angle.

Nous pouvons étudier l'effet de ce coefficient au moyen du programme suivant:

```
10 INPUT K
20 SCREEN 2
30 CIRCLE (127,95),50,1,,,K
40 GOTO 40
```

**PAINT** Cette instruction nous permet de peindre des surfaces en indiquant simplement un point quelconque dans une surface à colorier:

La forme générale est:

`PAINT (X,Y),` couleur de surface, couleur de ligne

avec:

| | |
|---|---|
| (X,Y) | coordonnées du point à indiquer |
| couleur de surface | couleur avec laquelle la surface doit être coloriée |
| couleur de ligne | couleur de la ligne continue délimitant la surface à colorier |

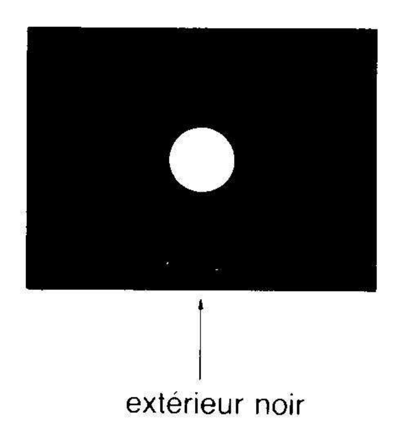
Exemple:

```
10 INPUT X,Y
20 SCREEN 2
30 CIRCLE (127,95),50,1,,,1.4
40 PAINT (X,Y),1
50 GOTO 50
```

cercle noir

Quand nous indiquons avec PAINT un point à l'intérieur du cercle, nous obtenons le résultat suivant:

Nous voyons comment la surface intérieure du cercle est coloriée. Cependant, quand nous indiquons un point hors du cercle, nous obtenons




Une remarque à ce sujet: dans les modes écran 5, 6, 7 et 8, la couleur de la ligne peut différer de la couleur de surface.

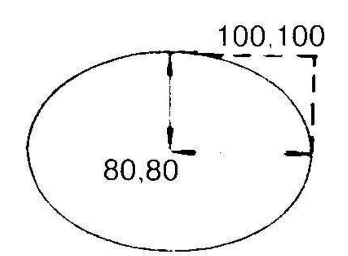
Exemple:

```
10 SCREEN 5
20 CIRCLE (127,95),50,1,,,1.365
30 PAINT (127,95),8,1
40 GOTO 40
```

**COPY**

COPY est une instruction très puissante qui permet de copier. Voici deux possibilités: soit nous effectuons une copie d'une partie de l'écran sur une autre partie de l'écran, soit nous effectuons une copie de l'écran pour la stocker dans un tableau. A noter que COPY ne peut être utilisé que dans les modes écran de 5 à 8.

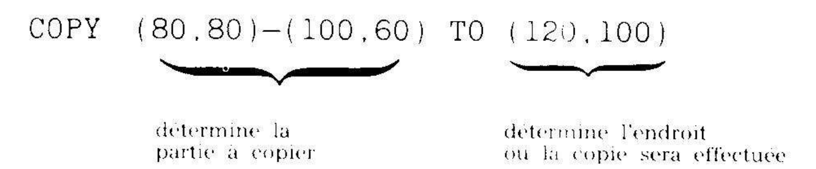
Prenons un exemple :



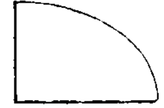

```
10 SCREEN 5
20 CIRCLE (80,80),20,15
30 PAINT (80,80),1,15
40 COPY (80,80)-(100,60) TO (120,100)
50 GOTO 50
```

L'écran reproduit à présent un disque aplati et un quart de disque

Les lignes 10, 20 et 30 ne posent aucun problème. La ligne 40 contient l'instruction COPY, dont la structure est la suivante:



Nous constatons que la partie à copier est indiquée immédiatement après COPY au moyen de deux points (X1,Y1) et (X2,Y2) qui déterminent un carré. Ceci est illustré dans la figure reproduite ci-dessus.



120,100

Nous voyons que l'endroit où la copie doit être reproduite n'est indiqué qu'au moyen d'un seul point.

Dans l'exemple suivant, nous montrons comment stocker une copie dans un tableau et comment réintroduire cette image sur l'écran à partir du tableau. Le stockage d'une partie de l'image dans un tableau entraîne immédiatement la question 'quelle doit être la dimension du tableau?' Cette dimension peut être déterminée au moyen de la formule suivante, qui peut être reprise telle quelle dans le programme:

```
INT((<PIXEL>*ABS(X2-X1)+1)*(ABS(Y2-Y1)+1)+7)/8)+4
```

Dans cette formule, <PIXEL> est égal à 4 dans les modes écran 5, 7 et 8 et égal à 2 dans le mode écran 6.

Bien que cette formule semble compliquée, il nous suffit de savoir qu'elle peut être reprise telle quelle dans le programme sans devoir nécessairement la comprendre. Pour notre exemple, X1=80, X2=100, Y1=80 et Y2=60. Notre programme se présente comme suit:

```
 5 B=INT((4*(ABS(100-80)+1)*(ABS(60-80)+1)+7)/8)+4
 6 DIM A(B)
10 SCREEN 5
20 CIRCLE (80,80)20,15
30 PAINT (80,80)1,15
40 COPY (80,80)-(100,60) TO A
50 COPY A,3 TO (120,120)
60 GOTO 60
```

La ligne 5 reproduit la formule dont il est question ci-dessus. A la ligne 6 figure la déclaration du tableau. La ligne 40 montre l'instruction COPY appropriée. Remarquez qu'après TO on ne mentionne que le nom du tableau. Enfin, la ligne 50 montre la forme selon laquelle l'image est placée sur l'écran à partir du tableau. Remarquez qu'un parametre 3 figure après la lettre A. Il s'agit du parametre d'orientation qui indique la manière dont l'image est placée. Pour une description complète à ce sujet, consultez l'aperçu des commandes MSX-BASIC.

# 4 SPRITES

## Sprite

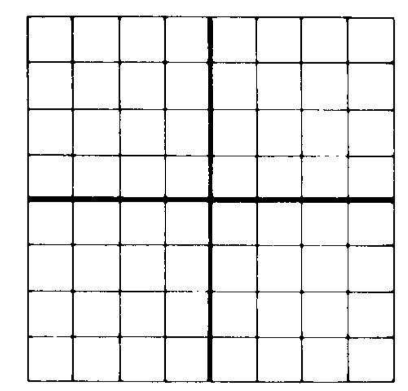
La toute première question que nous nous posons dans ce chapitre est naturellement 'qu'est-ce exactement qu'un sprite?' Un sprite est une figure que nous pouvons définir au moyen d'un matrice à points. Ensuite, nous donnons un numéro à cette figure. Au moyen d'une instruction spéciale, nous pouvons faire toutes sortes de manipulations sur cette figure. Ce qui est important, c'est de pouvoir déplacer facilement un sprite sur l'écran.

Evidemment, nous pouvons définir toutes sortes de figures avec les instructions PRINT et PSET. Cependant, il faut beaucoup d'instructions pour déplacer de telles figures ce qui est assez fastidieux.

Travailler avec des sprites offre donc beaucoup plus de souplesse. C'est un outil puissant pour animer à l'écran des êtres comme pacman.

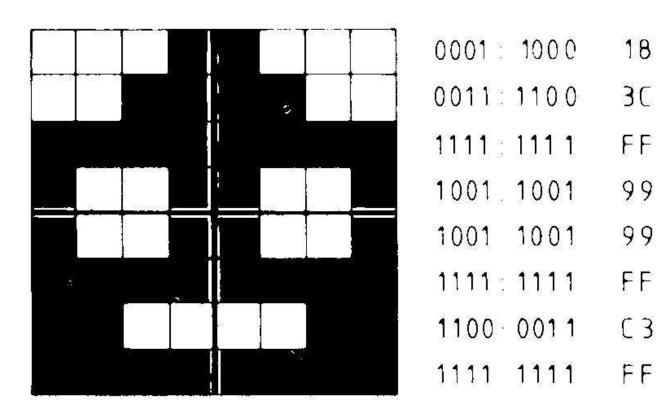
## Définir un sprite

Nous montrons maintenant comment définir un sprite. Nous partons d'une matrice de 8x8 points.



Ensuite, nous colorions en noir certaines cases pour obtenir la figure désirée.

Pour obtenir un petit revenant, nous pouvons colorier des cases de la manière suivante:




A gauche, nous voyons le petit revenant, et, à droite comment nous pouvons définir cette même figure au moyen de uns et de zéros (noir et blanc).

La ligne pointillée est une ligne auxiliaire, qui divise chaque ligne en deux séries de 4 uns et/ou zeros. La ligne supérieure comprend les séries 0001 et 1000, (code binaire).

Dans la colonne droite, nous voyons une annotation utilisant un autre code. Ce code peut être tiré directement de la table suivante (remarque: il s'agit ici du code hexadécimal).

| BIN. | HEX. | BIN. | HEX. | BIN. | HEX. | BIN. | HEX. |
|---|---|---|---|---|---|---|---|
| 0000 | 0 | 0100 | 4 | 1000 | 8 | 1100 | C |
| 0001 | 1 | 0101 | 5 | 1001 | 9 | 1101 | D |
| 0010 | 2 | 0110 | 6 | 1010 | A | 1110 | E |
| 0011 | 3 | 0111 | 7 | 1011 | B | 1111 | F |

Au moyen des codes trouvés dans cette table, nous définissons le sprite, dans un programme par l'instruction suivante:

SPRITE$ (numéro)= série instructions CHR$ (...)

Pour notre exemple:

```
SPRITE$(1)=CHR$(&H18)+CHR$(&H3C)+CHR$(&HFF)+
           CHR$(&H99)+CHR$(&H99)+CHR$(&HFF)+
           CHR(&HC3)+CHR$(&HFF)
```

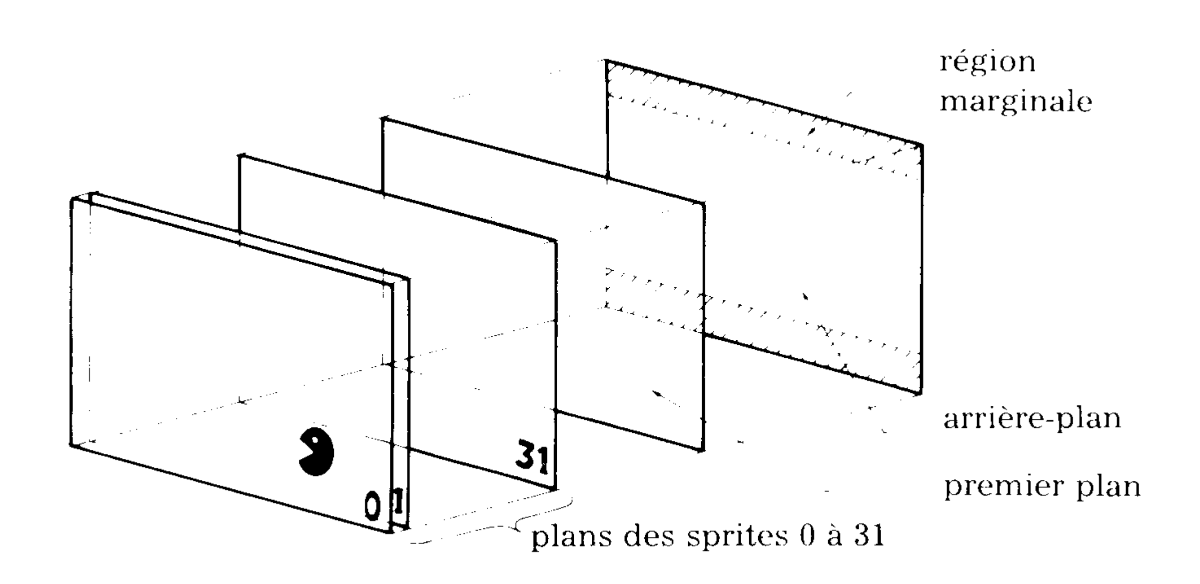
Remarquez qu'ici, les codes de série 18,3C,FF, etc. sont inclus dans les instructions CHR$, chaque code étant précédé par les signes &H.

**PUT SPRITE** Notre sprite (numéro 1) a été entièrement défini au moyen de la méthode que nous venons de décrire, et maintenant, nous pouvons placer cette figure n'importe où sur l'écran. A cette fin nous utilisons l'instruction PUT SPRITE, qui a la forme suivante:

PUT SPRITE plan, (x,y), couleur, numéro de sprite

Dans cette instruction:

plan — Le plan est un numéro entre 0 et 31. Il faut penser à la structure suivante:



Les sprites se déplacent pour ainsi dire sur des écrans en verre transparent placés les uns derrière les autres comme montré ci-dessus. La position d'un plan détermine quels sprites sont couverts quand deux sprites se chevauchent.

(x,y) les coordonnées de la position ou le sprite doit être représenté.

couleur le nombre indiquant la couleur du sprite.

numéro le numéro attribué au sprite au moyen de l'instruction SPRITE$ (numéro).

**ON SPRITE GOSUB et SPRITE ON** Dans beaucoup de jeux, il est important de savoir si, à un moment donné, deux sprites se rencontrent ou, pour utiliser une expression populaire, s'il se produit une collision.

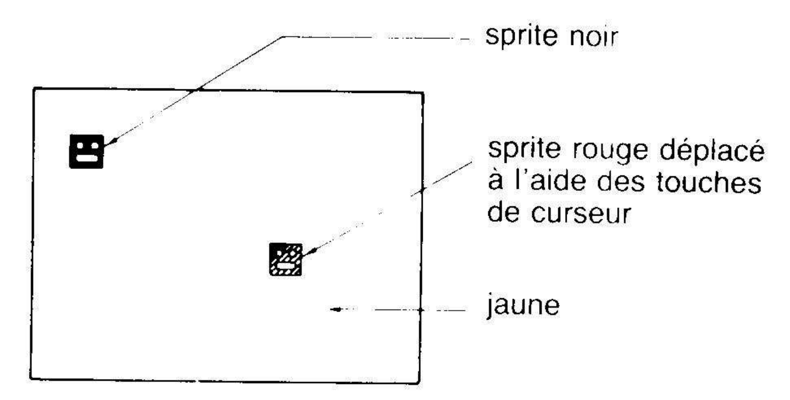
Pour constater cela, MSX-BASIC offre l'instruction ON SPRITE GOSUB. Celle-ci a toujours la forme suivante:

```
ON SPRITE GOSUB numéro de ligne
```

Quand l'ordinateur constate que deux sprites se recontrent, il effectue un saut vers le sous-programme indiqué par le numéro de ligne.

L'ordinateur n'effectue ce saut que lorsque l'instruction SPRITE ON lui a été indiquée.

**Exemple** Le programme suivant en donne un exemple.Deux revenants apparaissent à l'écran. L'un (noir) est stationnaire, tandis que l'autre (rouge) est commandé au moyen du clavier.
En image:




Voici le programme complet:

```
10 CLS
20 COLOR,11,11
30 SCREEN 2
40 X=100:'Y=100
```

```
 50 FOR K=1 TO 2
 60 SPRITE$(K)=CHR$(&H18)+CHR$(&H3C)+CHR$(&HFF)+
               CHR$(&H99)+CHR$(&H99)+CHR$(&HFF)+
               CHR$(&HC3)+CHR$(&HFF)
 70 NEXT K
 80 SPRITE ON
 90 PUT SPRITE 0,(40,40),1,1
100 D=STICK(0)
110 IF D=1 THEN Y=Y-1
120 IF D=2 THEN X=X+1:Y=Y-1
130 IF D=3 THEN X=X+1
140 IF D=4 THEN X=X+1:Y=Y+1
150 IF D=5 THEN Y=Y+1
160 IF D=6 THEN X=X-1:Y=Y+1
170 IF D=7 THEN X=X-1
180 IF D=8 THEN X=X-1:Y=Y-1
190 PUT SPRITE 1,(X,Y),6,2
200 ON SPRITE GOSUB 220
210 GOTO 100
220 PLAY "CCC"
230 RETURN
```

Les lignes 10 à 30 n'ont pas besoin d'explications. La ligne 40 détermine la position de départ du sprite rouge. Ensuite, les lignes 50 à 70 définissent deux sprites, qui ont d'ailleurs exactement la même forme (à savoir la forme déjà décrite).

Après, l'ordinateur fait particulièrement 'attention aux collisions' au moyen de l'instruction SPRITE ON (ligne 80). La ligne 90 met le sprite noir en position (40,40). Ensuite nous voyons la structure pour lire la position des touches du commande du curseur. Cette position est communiquée alors à l'instruction PUT SPRITE à partir de la ligne 190. Ainsi, le sprite rouge est placé.

La ligne 200 montre 'l'essai de collision'; si une collision se produit, l'ordinateur saute au sous-programme sonore de la ligne 220.

**Couleur du sprite (lutin)**

Nous avons vu dans la rubrique consacrée à PUT SPRITE la manière d'indiquer la couleur d'un sprite. Le MSX2 permet également à l'utillisateur de définir des sprites multicolores. A cette fin, nous utilisons les instructions COLOR SPRITES$ ou COLOR SPRITE. Seule la dernière instruction utilisée détermine la couleur visible sur l'écran. L'aperçu des commandes MSX-BASIC fournit une explication détaillée concernant COLOR SPRITE et COLOR SPRITE$, ainsi que des exemples.

**Remarques** Faisons quelques remarques au sujet des sprites:

- nous pouvons agrandir le sprite par 2 en remplaçant la ligne 30 par SCREEN 2,1

- nous pouvons également définir les sprites dans une matrice de 16x16. Cette matrice consiste alors en 4 blocs d'une matrice de 8x8. Ces blocs sont numérotés de la façon suivante.

| | |
|---|---|
| 1 | 3 |
| 2 | 4 |

La manière la plus simple est alors de fixer ce dessin en 4 chaînes:

```
A$=CHR$( )+ etc.      (définition bloc 1)
B$=CHR$( )+ etc.      (définition bloc 2)
```

et également C$ et D$ et ensuite

```
SPRITE$(1)=A$+B$+C$+D$
```

Quand nous travaillons avec des sprites 16x16, nous utilisons l'instruction SCREEN 2,2 ou SCREEN 2,3 quand nous désirons un agrandissement par 2.

- nous pouvons définir un maximum de 256 sprites avec une matrice 8x8 et un maximum de 64 sprites avec une matrice 16x16. Cependant, nous ne pouvons mettre qu'un seul sprite à chaque plan.

Enfin, signalons encore que dans les modes écran 1 à 4 on ne peut placer qu'un maximum de 4 sprites les uns à côté des autres. Dans les autres modes graphiques on peut placer un maximum de 8 sprites les uns à côté des autres.

# 5 FICHIERS, DISQUE MÉMOIRE, PUCE HORLOGE

**Introduction**

Un fichier (en anglais: file) est une appellation désignant un ensemble de données groupées. Ces données groupées peuvent être un certain nombre d'adresses (un fichier adresses), par exemple, mais aussi un programme en BASIC ou en code machine. Quel que soit son contenu, le fichier porte un nom unique permettant de l'utiliser dans des instructions ou commandes. On peut comparer un fichier à un livre dans une bibliothèque. Chaque livre contient une certaine quantité de données, qui peut donc faire l'objet d'un fichier. Le titre du livre correspondrait alors au nom du fichier.

A quoi servent les fichiers ?

Si nous disposons d'un lecteur de disquette, nous pouvons enregistrer nos programmes sur disquette sous forme de fichiers.

Mais ce n'est pas la seule possibilité qu'offre le MSX en matière de fichiers.

Les différentes possibilités qu'offre votre système sont résumées ci-après. Nous décrivons ci-dessous les peripheriques susceptibles de stocker des fichiers:

**CAS** le magnétophone à cassette.

Le magnétophone à cassette peut servir d'appareil de stockage peu coûteux pour conserver des programmes et données.

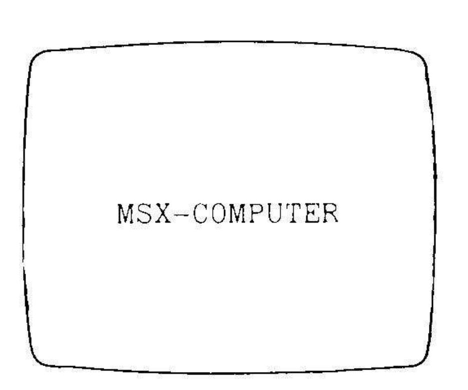
**GRP** l'écran graphique.

L'écran peut également être considéré comme un mode de stockage de fichiers. Il est toutefois évident que l'écran ne peut s'utiliser de la même manière qu'un magnétophone. Pourtant, il faut savoir que nous pouvons utiliser GRP dans certaines instructions pour reproduire du texte sur un écran graphique.

Vous trouverez un exemple simple ci-dessous. Nous verrons plus loin ce que signifient les diverses instructions spécifiques contenues dans ce programme.

```
10 SCREEN 2
20 OPEN "GRP:TEST" FOR OUTPUT AS#1
30 A$="MSX-COMPUTER"
40 PRESET (50,50)
50 PRINT #1,A$
60 GOTO 60
```

Après avoir donné la commande RUN, l'écran se présente comme suit:




Bref, nous voyons du texte (à savoir : MSX-COMPUTER) sur un écran graphique (SCREEN 2) !

**CRT** l'écran texte

Tout comme pour l'écran graphique, nous pouvons considérer l'écran texte comme un mode de stockage des fichiers.

**LPT** l'imprimante.

L'imprimante est un appareil périphérique permettant de 'matérialiser' les fichiers.

Vous trouverez une description detaillée de l'imprimante à l'annexe B.

**COM** l'interface RS232C.

Cette interface vous permet de connecter votre ordinateur à un autre ordinateur, par exemple, pour transmettre et recevoir des données. Au lieu de transmettre et recevoir, on dit aussi 'communiquer'. Voilà pourquoi on utilise la désignation COM.

Vous trouverez une description détaillée à ce sujet à l'annexe E.

**A à F** Unité de disquette A à F.

Lorsque nous indiquons dans des instructions-fichiers spécifiques une des lettres A à F, cette lettre sert à désigner une unité de disquette. L'unité de disquette est le système de stockage le plus professionnel. C'est le seul système permettant d'accéder à des données de manière aléatoire.

**MEM** Le disque mémoire.

Une partie de la mémoire est réservée pour servir de 'disque mémoire'. Cela signifie que nous pouvons utiliser cette partie de la mémoire comme s'il s'agissait d'une unité de disquette.

**Le nom d'un fichier**

Le nom d'un fichier peut se composer de 8 caractères au maximum (sauf avec le magnétophone à cassette: dans ce cas, 6 au maximum !). Exemples de nom:

```
TEST, TEST1 et EXP1
```

Outre les lettres et les chiffres, les signes suivants sont également admis:

```
$ & # % @ ( )_ \ ^ {} ~ et !
```

Après le nom, nous pouvons éventuellement placer un point et une extension (sauf avec le magnétophone à cassette). Cette extension nous permet d'indiquer le *type* de données (programme en BASIC, données générales, etc.) ayant trait au fichier.

Exemple :

```
TEST.BAS
```

Ce nom indique que le fichier TEST est un programme en BASIC.

Généralement, nous utilisons les extensions suivantes pour les fichiers sur disquette:

| *extension* | *signification* |
|---|---|
| ASC | fichier en code ASCII |
| ASM | programme d'assemblage |
| BAK | 'back up file' (copie d'un fichier) |
| BAS | programme BASIC |
| COM | programme utilitaire |
| TEX | fichier texte |
| PIC | fichier d'images |

NOTE: Les noms suivants ne peuvent pas être utilisés pour les extensions: AUX, CON, LST, PRN et NUL).

Parfois on désire assigner une commande non pas a seul fichier mais un ensemble de fichiers. On utilise alors des termes de substitution: les 'wild cards'. Ainsi

```
TEST.*
```

signifie: 'tous les fichiers portant le nom de TEST et ayant une extension quelconque'. Quand nous indiquons:

```
*.BAS
```

signifie: 'tous les fichiers ayant l'extension BAS'. Encore un exemple:

```
TEST*.BAS
```

signifie: 'tous les fichiers ayant l'extension BAS, dont les premiers caractères sont TEST'.

En plus du caractère * on peut utiliser également le point d'interrogation comme wild card. Ainsi l'indication de fichier

```
TEST?
```

concerne tous les fichiers dont les 4 premières lettres sont TEST; le cinquième caractère est arbitraire. Si nous avons par exemple la disposition des fichiers TEST1, TEST2, TEST3, tous ces fichiers sont indiqués par TEST?

**Les instructions OPEN, PRINT# et CLOSE.**

Lorsque nous souhaitons travailler avec un fichier, nous devons donner au préalable un nom et un numéro à ce fichier. Pour ce faire, nous utilisons l'instruction OPEN. Ensuite, nous remplissons le fichier séquentiel au moyen d'instructions PRINT#. Enfin, lorsque nous travaillons dans un programme avec fichiers, nous devons toujours terminer le programme au moyen d'une instruction CLOSE.

Ce qui précède est une simple énumération des instructions nécessaires pour désigner un fichier et lui attribuer des données.

Pour l'utilisateur débutant, c'est loin d'être clair. Nous allons donc illustrer en détail les instructions susmentionnées au moyen d'un exemple.
Exemple :

```
10 REM ECRIRE FILE
20 A$="SINGE"
30 B$="NOIX"
40 C$="MARIE"
50 OPEN "CAS:DATA" FOR OUTPUT AS#1
60 PRINT #1, A$
70 PRINT #1, B$
80 PRINT #1, C$
90 CLOSE #1
```

Les lignes 10 à 40 ne posent aucun problème. A la ligne 50, on trouve l'instruction OPEN. Celle-ci présente la forme suivante:

OPEN"nom du peripherique:nom du fichier"FOR OUTPUT AS numéro du fichier

Dans l'exemple qui précède, nous voyons que le mot CAS est utilisé pour indiquer le nom du périphérique. Cela signifie que nous souhaitons utiliser le magnétophone à cassette. Le nom du fichier est tout simplement DATA. La dernière partie:

```
FOR OUTPUT AS#1
```

signifie : 'nous allons transférer des données de la mémoire de l'ordinateur à la cassette" et en outre, 'nous donnons à ce fichier le numéro #1". Remarquons encore que toutes les données que nous enregistrons de cette manière sur la cassette formerons ensemble le fichier 'DATA' et que dorénavant ce fichier sera désigné par son numéro (#1).

Au lieu du mot OUTPUT, nous aurions également pu utiliser le mot INPUT. Dans ce cas, l'ordinateur comprend que l'on souhaite lire des données enregistrées sur le moyen de stockage; il s'agit donc de l'opération inverse. Nous en donnerons un exemple plus loin.

Les lignes 60 à 70 indiquent que les chaînes A$, B$ et C$ sont stockées dans le fichier au moyen de l'instruction PRINT#. Notons que le numéro de fichier se trouve immédiatement après le mot PRINT.

Lorsque toutes les données sont mémorisées, nous devons donner l'instruction CLOSE. Après le mot CLOSE, on note le numéro du fichier. La raison pour laquelle on donne l'instruction CLOSE est la suivante : l'acheminement des données de la mémoire de l'ordinateur vers le moyen de stockage s'effectue par l'intermédiaire d'une mémoire tampon. Sans l'instruction CLOSE, le contenu de la mémoire tampon pourrait ne pas être lu en entier.

Voici un autre exemple :

```
10 REM LIRE FILE
20 OPEN "CAS:DATA"FOR INPUT AS#1
30 INPUT #1, A$, B$, C$
40 PRINT A$, B$, C$
50 CLOSE #1
```

Résultat

```
SINGE     NOIX     MARIE
```

Nous constatons que la structure de ce programme est pratiquement identique à celle du précédent. A la place de OUTPUT dans le programme précédent, on trouve INPUT. Il s'agit donc de la lecture des données enregistrées sur le magnétophone à cassette. Au lieu d'instructions PRINT#, ce sont des instructions INPUT# que l'on utilise à présent; nous devons toutefois lire des données. L'instruction PRINT à la ligne 40 n'est reprise que pour nous assurer que les données initiales ont bien été mémorisées. Pour terminer, nous allons donner quelques exemples supplémentaires d'instructions OPEN sans reproduire un programme complet.

Exemple 1

```
OPEN "A:PROG.DAT" FOR INPUT AS#3
```

Il s'agit manifestement d'un fichier mémorisé sur la disquette A à partir duquel on va lire des données. Le numéro du fichier est 3, son nom est PROG.DAT.

Exemple 2

```
OPEN "GRP:TEST" FOR OUTPUT AS#1
```

Cette instruction a été utilisée dans le programme au début. L'information envoyée vers l'écran graphique.

L'instruction OPEN est expliquée de manière détaillée dans l'aperçu des commandes MSX.

**Manière dont les fichiers séquentiels sont stockés**

Le mot séquentiel signifie que les données sont enregistrées dans l'ordre, c'est-à-dire les unes après les autres. Nous expliquons ci-après les signes 'CR', 'LF' et ',', que l'on utilise comme signes de séparation. Notre illustration débutera par une explication de l'instruction PRINT.

**Au sujet de CR et LF**

L'abréviation CR indique retour au debut de la ligne et LF indique d'aller à la ligne suivante. CR et LF ont chacune leur code ASCII spécifique. Ces caractères sont utilisés aussi comme signe de séparation quand des données sont enregistrées sur cassette au moyen de l'instruction PRINT#.
Exemple 1

```
10 OPEN "CAS:ADDR" FOR OUTPUT AS#1
20 A=15:B=23.5:C=10:D=25.2:E 13
30 PRINT #1, A; B; C
40 PRINT #1, D; E
50 CLOSE #1
```

Dans ce cas, les données sur cassette peuvent être visualisées de la manière suivante

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 15 | , | 23.5 | , | 10 | CR | LF | 25.2 | , | 13 | CR | LF |

Il faut faire attention avec la manipulation de chaînes séparées au moyen de l'instruction PRINT#. Ici, une virgule doit toujours être utilisée comme signe de séparation.
Exemple 2

```
10 OPEN "CAS=NOM" FOR OUTPUT AS#1
20 A$="JEAN":B$="NICOLE"
30 PRINT #1, A$;",": B$
40 CLOSE #1
```

Exécution:

| | | | | |
|---|---|---|---|---|
| JEAN | , | NICOLE | CR | LF |

Remarques finales:

- le nombre de fichiers pouvant être ouverts simultanément est 1, à moins qu'un nombre plus grand soit spécifié au moyen d'une commande MAXFILES (par exemple MAXFILES=3).

Avec une disquette, on peut ouvrir 6 fichiers au maximum en même temps.

- au lieu de PRINT #, on peut utiliser aussi la forme PRINT USING #.

- au lieu de INPUT #, on peut utiliser aussi la forme LINE INPUT #.

**Fichier a accès sélectif**

Outre les fichiers séquentiels, le MSX permet aussi les fichiers à accès sélectif (Random Access files), mais uniquement sur disquette. On trouvera une description détaillée à ce sujet à l'annexe D.

**Disque Virtuel ou Disque Mémoire**

Le MSX BASIC n'utilise pas toute la mémoire pour le programme. Les concepteurs du système ont utilisé la place disponible pour permettre la création d'un disque mémoire (memory disk). De la sorte, vous pouvez utiliser une partie de la mémoire à la manière d'une unité de disquette. Cette possibilité est très pratique pour:

- stocker plus de programmes dans la mémoire;
- stocker des données dans la mémoire;
- effectuer certaines opérations, trier des données par exemple, à partir du disque mémoire plutôt que de la disquette, ce qui permet un gain de temps important.

Le disque mémoire ne peut être utilisé que pour stocker des programmes et des fichiers séquentiels. En d'autres termes : le disque mémoire ne permet pas de se servir de l'instruction COPY.

**Initialisation**

Pour utiliser le disque mémoire, on doit toujours placer au préalable l'instruction suivante dans le programme.

```
CALL MEMINI
```

Remarque importante: cette commande efface le contenu du disque mémoire (correspond à l'instruction CALL FORMAT pour un disquette).

Exemple de stockage d'un programme.

Supposons que le programme suivant se trouve en mémoire:

```
10 REM DEMO
20 END
```

et que nous souhaitons le stocker dans le disque mémoire.

Voici comment procéder :

a. Entrez la commande CALL MEMINI
b. Entrez la commande SAVE 'MEM:DEMO'

La commande CALL MFILES vous permet de constater que le programme est effectivement stocké.

Pour charger à nouveau le programme, on donne la commande LOAD 'MEM:DEMO'. Veillez à ce que le nom soit le même que celui figurant dans la commande SAVE. Il convient de *ne pas* répéter la commande CALL MEMINI pour stocker par la suite un autre programme dans le disque mémoire ou pour effectuer toute autre opération à partir du disque mémoire. Si vous le faites, la mémoire s'effacerait a chaque fois.

Les instructions/commandes suivantes sont utilisées avec le disque mémoire. Elles sont décrites de manière détaillée dans l'aperçu des commandes MSX-BASIC.

| | |
|---|---|
| CALL MEMINI | pour initialiser le disque mémoire. |
| CALL MFILES | donne le répertoire des fichiers. |
| CALL MKILL | pour effacer les fichiers. |
| CALL MNAME | pour donner un nouveau nom à un fichier. |

**La circuit d'horloge**

Votre ordinateur possède une circuit à part, dénommée circuit d'horloge qui présente la particularité de permettre le stockage de quelques données (et pas seulement l'heure), qui seront conservées même lorsqu'on éteint l'ordinateur.
On peut y stocker les données suivantes :

1. L'heure: instructions à utiliser: SET TIME et GET TIME. Pour la mise à l'heure, on utilise SET et pour la lire, on utilise GET. Ces instructions, accompagnées d'exemples, ainsi que celles qui suivent, sont décrites de manière détaillée dans l'aperçu des commandes MSX-BASIC.

2. La date: instructions à utiliser : SET DATE et GET DATE. Tout comme pour l'heure, la date est tenue à jour automatiquement.
La date peut avoir les formats suivants:
MM/JJ/AA ou JJ/MM/AA ou AA/MM/JJ
significant respectivement mois/jour/année ou jour/mois/année ou encore année/mois/jour, relais la version de votre MSX.
Avec le programme suivant vous pouvez trouvez le format de la date:

```
10 B=PEEK(&H2B)
20 A$="00000000"+BIN$(B)
30 C$=RIGHT$(A$,8)
40 IF LEFT$(C$,3)="000" THEN PRINT "AA/MM/JJ"
```

```
50 IF LEFT$(C$,3)="010" THEN PRINT "MM/JJ/AA"
60 IF LEFT$(C$,3)="010" THEN PRINT "JJ/MM/AA"
```

Dans les trois données qui suivent, seul le dernier nom que vous aurez introduit peut être conservé. A vous de choisir!

3. Mot de passe: l'instruction SET PASSWORD nous permet d'introduire un mot de passe dans le système pour en interdire l'utilisation aux personnes non autorisées. Si vous avez oublié le mot de passe introduit... vous pouvez malgré tout mettre le système en marche en appuyant simultanément sur les touches STOP et GRPH.

4. Prompt (message guide-opérateur): il s'agit du mot 'Ok' que vous voyez apparaître sur l'écran lorsque chaque commande a été exécutée. L'instruction SET PROMPT vous permet d'utiliser un autre mot.

5. Titre: l'instruction SET TITLE vous permet de faire apparaître le titre de votre choix sur l'écran lorsque vous mettez le système en marche.

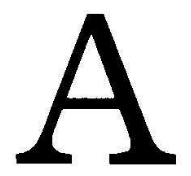
Les commandes suivantes se rapportent également au circuit d'horloge.

6. SET ADJUST: cette instruction vous permet de déplacer quelque peu l'image, par exemple si l'écran de votre téléviseur n'affiche pas la première colonne.

7. SET BEEP: cette instruction vous permet d'adapter la tonalité du 'bip' à votre convenance personnelle.

8. SET SCREEN: cette instruction vous permet d'initialiser la structure de l'image (voir SCREEN dans l'aperçu des commandes MSX-BASIC).

# ANNEXES



## UTILISATION DU MAGNÉTOPHONE À CASSETTE

Nous allons brièvement passer en revue la manière d'utiliser le magnétophone à cassette. Utilisez de préférence un magnétophone conseillé pour stocker des données. D'autres magnétophones à cassette peuvent poser des problèmes lorsqu'il s'agit de régler le volume ou la tonalité adéquats. La seule manière de s'en rendre compte, c'est d'en faire l'expérience !
Nous allons utiliser le programme de démonstration suivant :

```
10 REM DEMO CASSETTE
20 END
```

Voici comment procéder pour conserver ce programme sur cassette :

a. branchez le magnétophone.
b. introduisez le programme ci-dessus dans la mémoire de l'ordinateur.
c. mettez le magnétophone en position 'enregistrement' et donnez ensuite la commande CSAVE "DEMO".
Lorsque cette instruction est exécutée, vous verrez apparaître 'Ok' sur l'écran. Vous pouvez vérifier si l'enregistrement est correct au moyen de la commande CLOAD? "DEMO", en n'oubliant pas de mettre le point d'interrogation après CLOAD. Cette commande est donnée après avoir rebobiné la cassette et après avoir mis le magnétophone en position PLAY. Comme le programme initial se trouve toujours dans la mémoire de l'ordinateur, il est alors comparé avec l'enregistrement, si la comparaison est bonne l'ordinateur affiche OK.

Pour lire par la suite le programme enregistré sur la cassette, on utilise la commande :

```
CLOAD
```

L'ordinateur charge dans sa mémoire le premier programme qu'il 'rencontre'. Vous pouvez également donner un nom de

programme. Dans ce cas, la commande présente la structure suivante : CLOAD 'nom du programme'. Dans le cas de l'exemple ci-dessus, nous obtenons :

```
CLOAD"DEMO"
```

Quelques remarques pour terminer :

- Si un problème se présente lors de la lecture d'une cassette, il y a deux possibilités :

  - l'ordinateur affiche sur l'écran le message 'device I/O error'.
  - aucun message n'apparaît sur l'écran.

  Dans ce cas, rebobinez la bande et faites des essais en modifiant le volume et la tonalité de votre magnétophone pour trouver la position permettant de charger correctement. Généralement, vous pouvez amener le bouton du volume à 80 % du volume maximal et mettre le bouton de tonalité sur 'max'. Par ailleurs, n'utilisez que des cassettes ferro et non pas chrome.

- Le magnétophone à cassette peut aussi être utilisé pour stocker des fichiers séquentiels.

- Certaines bandes préenregistrées ne peuvent pas être copiées. Dans ce cas, votre magnétophone n'est pas en cause; ce sont les cassettes qui sont munies d'un système de protection !

**Aperçu des commandes**

Les commandes suivantes se rapportent à l'utilisation des cassettes. Vous en trouverez une description détaillée dans l'aperçu des commandes MSX-BASIC.

| | |
|---|---|
| BLOAD | pour charger un programme en langage machine. |
| BSAVE | pour mémoriser un programme en langage machine. |
| CLAOD | pour charger un programme. |
| CLOSE | pour fermer des fichiers. |
| CSAVE | pour mémoriser un programme. |
| INPUT# | pour lire un fichier séquentiel. |
| LINE INPUT# | pour lire un fichier séquentiel ligne par ligne. |
| LOAD | pour charger un programme en forme ASCII. |
| MERGE | pour fusionner des programmes. |
| OPEN# | spécifie la manière dont un fichier est utilisé. |
| PRINT# | pour introduire des données dans un fichier séquentiel. |
| PRINT# USING | pour introduire des données format est indiqué dans un fichier séquentiel. |
| SAVE | pour mémoriser un programme en forme ASCII. |

| | |
|---|---|
| **Aperçu des fonctions** | Ecrites de manière détaillée dans l'aperçu des commandes MSX-BASIC |
| | |
| EOF | fin d'un fichier. |
| INPUT$# | pour écrire des données alphanumériques. |
| LINE INPUT$# | pour écrire des données alphanumériques ligne par ligne. |
| VARPTR# | pour trouver l'adresse d'un bloc de contrôle de fichier. |

# B UTILISATION DE L'IMPRIMANTE

Sur votre ordinateur, vous pouvez brancher des imprimantes MSX ou non MSX. Une imprimante MSX est conçue spécialement pour les systèmes MSX au niveau des connexions, des codes de commande, du jeu de caractères. Par contre, des problèmes pouvent se présenter lorsqu'on utilise une imprimante non MSX.

Vous commencerez vraisemblablement par utiliser votre imprimante pour imprimer des listings de programme (commande LLIST). Vous pourrez également utiliser l'imprimante pour imprimer des données (instructions LPRINT). Enfin, vous pouvez stocker les données dans un fichier séquentiel.

**Aperçu des commandes**

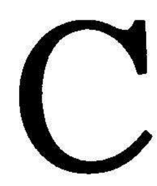
Les commandes énumérées ci-dessous se rapportent à l'utilisation de l'imprimante. Pour une explication détaillée, voir l'aperçu des commandes MSX-BASIC.

| | |
|---|---|
| CLOSE | pour fermer les fichiers. |
| LLIST | pour imprimer un programme. |
| LPRINT | pour imprimer des données. |
| OPEN | spécifie le mode d'utilisation d'un fichier. |
| PRINT# | pour imprimer des données en tant que fichier séquentiel. |
| PRINT# USING | pour imprimer des données en tant que fichier séquentiel, tout en déterminant le format. |

**Aperçu des fonctions**

Fonctions spécifiques (décrites de manière détaillée dans l'aperçu des commandes MSX-BASIC) :

| | |
|---|---|
| VARPTR# | pour trouver l'adresse du bloc de contrôle de fichier. |



# UTILISATION DES CONNECTEURS DE POIGNÉE DE JEU

Ces connecteurs sont désignés par la mention 'hand control' figurant sur l'ordinateur.

Ces connecteurs vous permettent de brancher une table traçante, un crayon lumineux, une souris ou un track ball.

**Aperçu des commandes**

Les commandes qui suivent se rapportent à l'utilisation des connecteurs de poignée de jeu. Pour une description détaillée, voir l'aperçu des commandes MSX-BASIC.

| | |
|---|---|
| ON STRIG GOSUB | vérifie si le bouton de tir de la poignée est enfoncé. |
| STRIG/ON/OFF/STOP | activent la commande du bouton de tir de la poignée. |

**Aperçu des fonctions**

Ces fonctions sont décrites de manière détaillée dans l'aperçu des commandes MSX-BASIC.

| | |
|---|---|
| PAD | donne l'etat d'une tablette graphique, d'un crayon optique, d'une souris ou d'un 'track ball'. |
| PDL | donne la position d'un contrôleur de pedale de jeu (paddle controller). |
| STICK | donne la position de la poignée de jeu. |
| STRIG | indique si le bouton de tir de la poignée de jeu est enfoncé. |

# D UTILISATION DE L'UNITÉ DE DISQUETTE

Si vous disposez d'une unité de disquette, vous pouvez vous en servir pour stocker vos données et programmes. De nombreux logiciels sont vendus sur disquette. Si l'on introduit un tel logiciel dans l'unité de disquette, le programme se charge généralement dès que l'on branche l'ordinateur.

Si vous ne souhaitez pas qu'un programme soit chargé automatiquement, branchez d'abord votre ordinateur et n'introduisez votre disquette dans l'unité de disquette que lorsque le message 'Ok' apparaît à l'écran. La commande FILES vous permet de faire apparaître le répertoire de tous les fichiers stockés sur la disquette. Ce point sera approfondi ci-dessous. A présent, nous allons traiter de l'utilisation de la disquette. Nous traiterons des commandes COPY (pour copier un fichier), FILES (pour afficher le répertoire des fichiers stockés sur la disquette) KILL (pour supprimer des fichiers). Ensuite, nous traiterons de l'instruction FORMAT (qui permet de préparer une disquette qui n'a jamais été utilisée), LOAD (pour charger en mémoire un fichier stocké sur la disquette) et SAVE pour sauvegarder un fichier aléatoire sur disquette.

Enfin, nous indiquons comment utiliser les fichiers à accès sélectif (Random Access files). Cette annexe se termine par une énumération des commandes/fonctions spécifiques concernant l'utilisation des disquettes.

## Les unités de disquette et leurs commandes

Le dispositif dans lequel on met une disquette, est appelé unité de disque. On peut connecter plus d'une unité à l'ordinateur et on peut donc se demander: 'comment indiquer de quelle unité il s'agit?'. Or, la réponse est simple: on indique les unités de manière symbolique, en utilisant des lettres. Quand on a deux unités, elles ont les noms A et B. On utilise ces lettres aussi pour indiquer dans quelle unité un certain fichier se trouve. Ainsi:

```
A:TEST.BAS
```

signifie: 'le programme TEST.BAS qui se trouve sur le disque de l'unité A'.

Voici un exemple d'une commande qui comporte des noms de fichiers et d'unités, à savoir la commande COPY. Ceci est la commande pour préparer des copies de fichiers.
La commande:

```
COPY "A:TEST.BAS" TO "B:"
```

signifie: 'copiez le programme TEST.BAS qui se trouve sur le disque dans l'unité A, vers le disque qui se trouve dans l'unité B'.

Autre exemple:

```
COPY "A:TEST.BAS" TO "B:EXP1.BAS"
```

veut dire: 'copiez le programme TEST.BAS et stockez-le sous le nom EXP1.BAS sur le disque se trouvant dans l'unité B'. Autre exemple, concernant la commande qui donne la liste des fichiers se trouvant sur un disque. Cette commande porte le nom FILES. Ainsi

```
FILES "A:"
```

signifie: 'Donnez la liste de tous les fichiers qui se trouvent sur le disque dans l'unité A'.

Avec la commande FILES nous pouvons indiquer aussi un seul nom, pour vérifier si ce fichier se trouve bien sur le disque, par exemple:

```
FILES "A:TEST.BAS"
```

Si ce fichier se trouve en effet sur le disque, le nom TEST.BAS est affiché encore une fois sur l'écran. Si ce n'est pas le cas, l'ordinateur nous signale:

```
File not found
```

ce qui veut dire que 'le fichier en question n'a pas été trouvé'. Encore un exemple, qui concerne l'effacement de fichiers. A cette fin on utilise la commande KILL. Ainsi

```
KILL "A:TEST.BAS"
```

signifie: 'effacez le programme TEST.BAS se trouvant sur le disque de l'unité A.'
que signifie l'exemple suivant:

```
KILL "A:*.*"
```

Ceci signifie: 'effacez tous les programmes se trouvant sur le disque de l'unité A.
On verra tout à l'heure comment utiliser les noms A et B quand on n'a qu'une unité à sa disposition.

**AUTOEXEC.BAS** Si un programme sur disquette (incluse dans l'unité A) est appellé AUTOEXEC.BAS, ce programme sera automatiquement exécuté lorsque le système sera mis en marche.

**FORMAT** Prenons un disque vierge, pour le formater. Ce traitement préliminaire est nécessaire pour pouvoir stocker des fichiers sur le disque. Nous introduisons:

```
_FORMAT
```

ou

```
CALL FORMAT
```

Entrez au clavier: ... et suivez les instructions qui s'affichent sur l'écran.

Quand le disque a été formaté, il affiche sur l'écran:

```
Format complete
Ok
```

Pour voir si nous avons effectivement un disque sur lequel on peut stocker des programmes, nous introduisons un court programme de démonstration:

```
10 PRINT "MSX DISK BASIC"
```

Pour stocker ce programme sur disque, on utilise la commande SAVE. On introduit la commande:

```
SAVE "A:TEST.BAS"
```

c'est-à-dire: 'stockez le programme sous le nom TEST.BAS sur le disque dans l'unité A'.

Quand la touche RETURN est enfoncée, l'unité tourne un instant. Est-ce que le programme est vraiment stocké sur le disque?

On peut le vérifier en effaçant le programme existant qui se trouve encore dans la mémoire, au moyen de la commande NEW (faites le effectivement!). Maintenant on introduit le programme à partir du disque, au moyen de la commande LOAD:

```
LOAD "A:TEST.BAS"
```

Quand la touche RETURN est enfoncée, l'unité tourne un instant. Si on donne alors la commande LIST, le programme stocké en effet dans la mémoire de l'ordinateur apparaitra sur l'écran.

Pour vérifier si le fichier TEST.BAS se trouve sur le disque de l'unité A, on aurait pu donner aussi la commande FILES déjà décrite, par exemple:

```
FILES A:*.*
```

(faites-le effectivement), ou simplement:

```
FILES
```

**On n'a qu'une seule unité à sa disposition**

Dans la plupart des cas on n'aura qu'une seule unité à sa disposition et on peut se demander: 'comment faire maintenant?'.

Or, dans ce cas les noms A et B ne concernent pas deux unités, mais deux disques. Nous l'expliquons au moyen de l'exemple suivant:

Admettons qu'on ait deux disques (A et B) et que l'on veuille copier le programme TEST.BAS du disque A sur le disque B. Supposons également que le disque A se trouve encore dans l'unité.

On donne maintenant la commande suivante:

```
COPY "A:TEST.BAS" TO "B:"
```

Le programme TEST.BAS est chargé dans la mémoire de l'ordinateur et ensuite l'ordinateur affiche:

```
Insert diskette for drive B
and strike a key when ready
```

Dans ce cas on devra mettre le disque B dans l'unité.

Si le programme ne peut pas être copié en une seule fois, cela devra se faire en plusieurs étapes. L'ordinateur indique toujours si le disque A ou le disque B est à mettre dans l'unité.

Encore une astuce pratique: écrivez le nom, donc A ou B, sur le disque. Ainsi, vous éviterez vraiment un tas de problèmes!

**Fichiers séquentiels**

MSX-Disk BASIC offre également la possibilité de travailler avec des fichiers séquentiels. Le principe d'un fichier séquentiel a deja été expliqué en Annexe A, il n'est donc plus étudié ici.

Cependant, il reste un point à étudier, à savoir la fonction APPEND. Cette fonction permet d'étendre un fichier séquentiel, ce qui n'est pas possible avec la cassette.

Quand on veut stocker davantage de données dans un fichier existant, autrement dit quand on veut étendre ce fichier, on donne l'instruction OPEN suivante:

```
OPEN "nom" FOR APPEND AS#numero
```

Supposons que nous ayons préparé un petit fichier avec le programme suivant:

```
10 OPEN "A:DAT" FOR OUTPUT AS#1
20 INPUT X$
30 IF X$="END" THEN CLOSE #1:END
40 PRINT #1, X$
50 GOTO 20
```

Quand on a stocké quelques chaînes dans A:DAT de cette façon, le fichier A:DAT est présent sur disque.

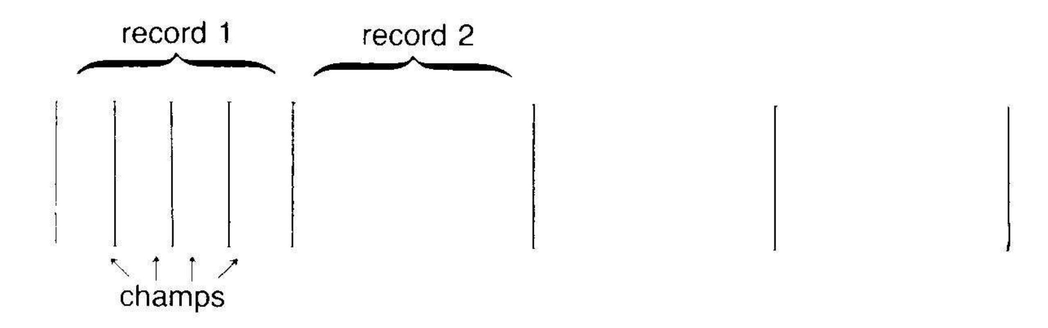
Le programme suivant montre alors comment on peut ajouter des données à ce fichier existant:

```
10 OPEN "A:DAT" FOR APPEND AS#1
20 INPUT X$
30 IF X$="END" THEN CLOSE #1:END
40 PRINT #1, X$
50 GOTO 20
```

La différence entre le programme précédent et celui-ci est le mot APPEND.

**Fichiers à accès aléatoire**

Un fichier à accès aléatoire est un fichier avec une propriété très spéciale, à savoir que l'on peut stocker ou effacer des données dans certaines positions du fichier. La figure suivante l'explique:




Cette figure montre une succession d'enregistrements, dont chacun est sous-divisé en des champs (FIELDS).

Cela implique que chaque enregistrement a une longueur fixe (LEN) en termes d'octets et que cet enregistrement consiste en des champs qui ont à leur tour une longueur fixe. Pour définir un tel fichier à accès sélectif, il faut d'abord spécifier LEN, par exemple.

```
OPEN "A:DATA" AS#1 LEN=34
```

Dans cet exemple chaque enregistrement a une longueur de 34 octets. Comme l'expression FOR INPUT, OUTPUT ou APPEND n'est pas inclue dans cette instruction OPEN, l'ordinateur sait automatiquement qu'il s'agit d'un fichier à accès aléatoire. En outre il faut indiquer la structure de chaque champ et également par quel nom on peut renvoyer à un tel champ. Par exemple:

```
FIELD #1, 20 AS A$, 8 AS D$, 4 AS S$, 2 AS I$
```

On voit dans l'exemple que chaque enregistrement consiste en 4 champs avec les noms A$, D$, S$ et I$, qui comprennent alors la quantité d'octets indiquée. Il va sans dire que le nombre total d'octets doit être égal au plus au nombre spécifié dans LEN (vérifiez si c'est en effet le cas!)

Ainsi le cadre du fichier a été fixé et on peut voir comment des données sont placées dans un tel fichier. A cette fin, on utilise les instructions LSET et RSET qui servent à positionner à gauche ou à droite une chaîne dans un champ (voir page 205).

L'exemple suivant montre les différentes possibilités:

```
LSET A$ = WA$
RSET D$ = MKD$ (WD#)
RSET S$ = MKS$ (WS!)
RSET I$ = MKI$ (WI%)
```

WA$ represente une chaîne, WD# un nombre en double préci-

sion, WS! un nombre en simple précision et WI% un nombre entier. Remarquez que chaque nombre doit toujours être converti en une chaîne au moyen de la fonction prévue à cette fin.

Ces valeurs doivent être maintenant mises dans le fichier... mais comment? Cela dépend de l'enregistrement dans lequel on veut mettre ces données. Cela se fait au moyen de l'instruction

```
PUT #1, numéro de l'enregistrement
```

par exemple

```
PUT #1,3
```

Attention. Après que toutes les données attribuées à A$, D$, S$ et I$ au moyen de LSET et de RSET, aient été stockées dans le fichier, on peut attribuer de la même façon de nouvelles données à A$, D$, S$ et I$, lesquelles données peuvent alors être stockées dans un autre enregistrement.

Enfin, un fichier doit toujours être fermé à la fin d'un programme.

```
CLOSE #1
```

Pour la lecture, le fichier à accès aléatoire est défini de la manière déjà décrite.

Quand on a, avec le même fichier des actions de sortie et d'entrée dans un seul programme, le fichier ne doit être défini qu'une seule fois. L'introduction se fait au moyen de l'instruction GET. Ainsi

```
GET #1,3
```

attribuera le contenu de l'enregistrement 3 à A$, D$, S$ et I$. Ces valeurs peuvent être affichées avec:

```
PRINT A$
PRINT CVD (D$)
PRINT CVS (S$)
PRINT CVI (I$)
```

remarquez les instructions CVD, CVS et CVI qui permettent de transformer des nombres donnés sous forme de chaîne en nombres arithmétiques.

**Aperçu des commandes**

Les commandes suivantes se rapportent à l'utilisation des disquettes. Vous en trouverez une description détaillée dans l'aperçu des commandes MSX-BASIC

BLOAD pour charger un programme en langage machine ou la mémoire vidéo.

BSAVE pour mémoriser un programme en langage machine ou la mémoire vidéo.

CLOSE pour fermer des fichiers.

| | |
|---|---|
| COPY | pour copier des fichiers ou des parties de l'écran. |
| DSKO$ | pour placer des données dans un secteur spécifié. |
| FIELD | pour avoir accès au tampon utilisé dans le cas d'un fichier à accès aléatoire. |
| FILES | affiche le répertoire des fichiers conservés sur la disquette. |
| LFILES | idem, mais le répertoire sont imprimés par l'imprimante. |
| FORMAT | pour formater une disquette. |
| GET | pour écrire un enregistrement dans le tampon d'un fichier à accès aléatoire. |
| INPUT# | pour lire un fichier séquentiel. |
| KILL | pour effacer un fichier. |
| LINE INPUT# | pour lire un fichier séquentiel ligne par ligne. |
| LOAD | pour charger un fichier. |
| LSET, RSET | utilisé avec les fichiers à accès sélectif. |
| MAXFILES | indique le nombre maximal de fichiers pouvant être ouverts. |
| MERGE | pour fusionner des programmes. |
| NAME | pour attribuer un nouveau nom à un fichier. |
| OPEN | spécifie la manière dont un fichier est utilisé. |
| PRINT# | pour introduire des données dans un fichier séquentiel. |
| PRINT# USING | pour introduire des données dont le format est indiqué dans un fichier séquentiel. |
| PUT | utilisé avec les fichiers à accès aléatoire. |
| SAVE | pour stocker un fichier sur disquette. |
| CALL SYSTEM | pour retourner au système MSX-DOS. |

**Aperçu des fonctions** Décrites de manière détaillée dans l'aperçu des commandes MSX-BASIC:

| | |
|---|---|
| CVI, CVS, CVD | pour convertir des chaînes en nombres. Utilisées avec les fichiers à accès aléatoire. |
| DSKF | détermine la place restante sur une disquette |
| DSKI$ | pour charger un secteur. |
| EOF | fin d'un fichier. |
| INPUT$ | pour écrire des données alphanumériques. |
| LINE INPUT$# | pour écrire des données alphanumériques ligne par ligne. |
| LOC | donne la dernière position dans le fichier indiqué. |
| LOF | indique la longueur d'un fichier. |
| MKI$, MKS$, MKD$ | voir CVI, CVS et CVD. Pour convertir un nombre en une chaîne. |
| VARPTR# | pour trouver l'adresse d'un bloc de contrôle de fichier. |

# E

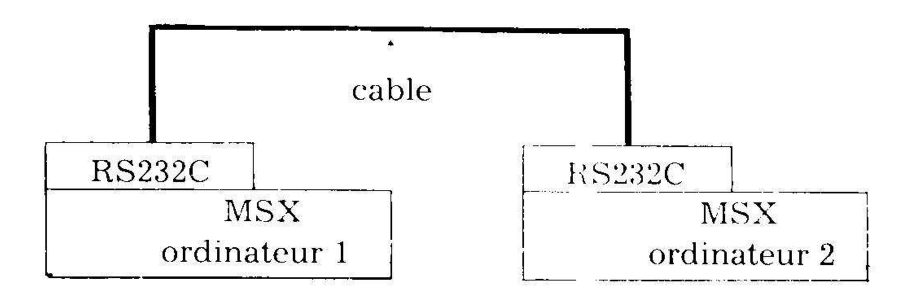
# UTILISATION DE L'INTERFACE RS232C

La communication est une notion importante dans le monde de l'informatique. Par cette notion on entend la transmission de données d'un endroit à un autre. Pensons à deux ordinateurs connectés l'un à l'autre. N'est-il pas alors très utile de pouvoir transmettre de manière simple des programmes ou des fichiers de données?

Or, pour permettre cette transmission, une interface RS232C a été mise au point pour l'ordinateur MSX. Cet appareil sert à transmettre de l'information sous une forme spécialement appropriée.

La RS232C est une interface série, parce que les uns et les zéros, avec lesquels toutes les données sont enregistrées, sont transmis l'un après l'autre, donc en série.

Pour laisser ce procédé se dérouler convenablement, il faut évidemment des signaux de contrôle. Nous montrons ci-dessous comment deux ordinateurs MSX, équipés chacun d'une interface RS232C, peuvent communiquer l'un avec l'autre. Schématiquement:




Dans cet exemple un ordinateur MSX est en communication avec un autre ordinateur MSX, mais cet autre ordinateur pourrait être un autre appareil équipé d'une interface RS232C (pensez par exemple à un modem pour communiquer avec un autre ordinateur par une ligne téléphonique).

L'interface elle-même consiste en une cartouche sans câble. Sur cette cartouche se trouve une fiche (connecteur Centronics à 25 broches).

Des câbles sont disponibles pour raccorder les cartouches des deux ordinateurs (voir la figure).

Introduisons le programme suivant sur l'ordinateur 1:

```
10 PRINT "RS232C"
20 PRINT "MSX"
```

On donne maintenant la commande suivante:

```
CALL COMINI
SAVE "COM:"
```

L'intention est de sauvegarder ce programme par l'indication COM, qui concerne l'interface RS232C.
Quand on introduit sur l'ordinateur 2:

```
CALL COMINI
LOAD "COM:"
```

Le programme est transmis quand les touches RETURN des deux ordinateurs sont enfoncées.

Attention: quand on attend un trop long temps après avoir donné la commande LOAD, il s'affiche un message d'erreur (Device I/O error). La raison en est que par l'introduction de la commande SAVE l'utilisateur indique que l'ordinateur 1 désire transmettre des données et l'ordinateur 1 regarde si l'ordinateur 2 est 'prêt' à recevoir ces données (ce qui est indiqué à l'aide de la commande LOAD). Si cela dure trop longtemps, un message d'erreur est affiché.

En réalité, cela s'applique à toute communication: si l'appareil récepteur n'indique pas qu'il est 'prêt à recevoir des données', la communication est coupée automatiquement.

L'exemple suivant montre comment un message est transmis de l'ordinateur 1 à l'ordinateur 2.
Sur l'ordinateur, on introduit le programme suivant:

```
10 OPEN "COM:" FOR OUTPUT AS#1
20 INPUT A$
30 PRINT #1, A$
40 CLOSE #1
```

Sur l'ordinateur 2 le programme suivant est introduit:

```
10 OPEN "COM:" FOR INPUT AS#1
20 INPUT #1, A$
30 PRINT A$
40 CLOSE #1
```

Si l'on introduit sur l'ordinateur 1 la chaîne MSX, celle-ci apparaît également sur l'écran de l'ordinateur 2 (si la commande RUN a été donnée dans le programme de l'ordinateur 2).

Encore un exemple:
Si la commande

```
CALL COMTERM
```

est donnée sur les deux ordinateurs, tout ce qu'on introduit sur l'ordinateur 2, s'affiche également à l'ordinateur 1. Dans ce cas on a fait des deux ordinateurs un 'terminal' esclave (machine à écrire pour transmettre des données), ce qui permet d'échanger des messages d'une manière simple.

**Raccordement à un appareil autre qu'un ordinateur MSX (par exemple un ordinateur d'une autre marque)**

Voilà un sujet qu'on ne peut pas étudier en général, à cause du fait qu'il faut régler toutes sortes de choses comme:
- la vitesse de transmission
- le contrôle de parité
- la longueur des mots
- le nombre de bits d'arrêt

et d'autres choses encore.

En general la communication entre deux ordinateurs, a l'aide de l'interface RS232C, necessite certaines connaissances techniques et présente des difficultes pour l'utilisateur débutant. Une étude approfondie des paramètres de communications du chacun des ordinateurs sera alors necessaire.

**Aperçu des commandes**

Les commandes suivantes se rapportent à l'utilisation d'une interface RS232C. Vous en trouverez une description détaillée dans l'aperçu des commandes MSX-BASIC.

| | |
|---|---|
| CALL COMINI | pour l'initialisation de l'interface RS232C (nombre de bauds, longueur des 'portions' envoyées, etc.). |
| CALL COMON<br>CALL COMOFF<br>CALL COMSTOP<br>CALL COM | ces instructions sont utilisées pour exécuter automatiquement un sous-programme lorsqu'une commande est reçu via l'interface RS232 |
| CALL COMTERM | lance le mode émulateur du terminal (terminal emulator mode). |
| CALL COMBREAK | pour transmettre des caractères d'interruption (break characters). |
| CALL COMDTR | pour mettre temporairement hors circuit l'interface RS232C. |
| CALL COMSTAT | pour trouver ce qui s'est passé en cas d'erreur. |
| CLOSE | pour fermer la connexion RS232C. |
| INPUT# | pour recevoir des données au moyen de l'interface RS232C. |
| LOAD | pour recevoir un programme au moyen de l'interface RS232C. |
| LINE INPUT# | pour recevoir des données ligne par ligne au moyen de l'interface RS232C. |
| MERGE | pour fusionner des programmes au moyen d'interface RS232C. |
| OPEN | pour ouvrir la connexion RS232C. |
| PRINT# | pour transmettre des données au moyen de l'interface RS232C. |
| PRINT# USING | pour transmettre des données au moyen de l'interface RS232C, lorsque le format est déterminé. |
| SAVE | pour transmettre un programme au moyen de l'interface RS232C. |

**Aperçu des Fonctions** Décrites de manière détaillée dans l'aperçu des commandes MSX BASIC :

| | |
|---|---|
| EOF | fin du fichier transmis au moyen de l'interface RS232C. |
| INPUT$# | pour recevoir des données alphanumériques au moyen de l'interface RS232C. |
| LINE INPUT$# | pour recevoir des données alphanumériques ligne par ligne au moyen de l'interface RS232C. |
| VARPTR# | pour trouver l'adresse du bloc de contrôle de fichier. |



# SIGNES ET EXPRESSIONS SPÉCIAUX

## Signes d'opération

La table suivante donne un aperçu des signes que nous pouvons utiliser pour les opérations mathématiques. Le dernier chiffre indique la valeur de priorité: les opérations sont exécutées d'après ces valeurs de priorité (allant du chiffre le plus bas au chiffre le plus élevé). En cas d'équivalence de priorité, l'expression est calculée de la gauche vers la droite.

| *signe* | *signification* | *exemple* | *priorité* |
|---|---|---|---|
| + | addition | 3+5 | 6 |
| - | soustraction | 5-3 | 6 |
| * | multiplication | 5x3 | 3 |
| / | division | 5/3 | 3 |
| ^ | exponentielle | $2^3$ | 1 |
| − | change le signe | −3 | 4 |
| MOD | reste dans division à nombre entier | 12 MOD 3 | 5 |

## Signes dans des expressions logiques

Les symboles suivants peuvent figurer dans des expressions. Il est vérifié alors s'ils sont vrais ou non (par exemple après le terme IF)

| *signe* | *signification* | *exemple* |
|---|---|---|
| = | est égal à | IF A=B THEN... |
| < | inférieur à | IF A<B THEN... |
| > | supérieur à | IF A>B THEN... |
| <> ou >< | différent de | IF A<>B THEN... |
| <= ou =< | inférieur ou égal à | IF A<=B THEN... |
| >= ou => | supérieur ou égal à | IF A>=B THEN... |

## Termes pour combiner des expressions logiques

Des expressions comme A=B et C<>D peuvent être combinées avec des termes spécifiques.

La table suivante donne les termes différents avec les tables de vérité. Dans ces tables, 1 indique que l'expression est vrai et 0 que l'expression est fausse. Ainsi, on voit par exemple que le terme AND

U1 **AND** U2

est 1 si U1 et U2 sont égales à 1. Par U1 et U2, sont alors expressions comme A=B et C<>D. La table indique dans ce cas que l'expression entière est vraie (1) si U1 et U2 sont vraies aussi (1).

| *terme* | | table de vérité | |
|---|---|---|---|
| **NOT** | U1 | | NOT U1 |
| (négation) | 1 | | 0 |
| | 0 | | 1 |
| **AND** | U1 | U2 | U1 AND U2 |
| (produit logique) | 1 | 1 | 1 |
| | 1 | 0 | 0 |
| | 0 | 1 | 0 |
| | 0 | 0 | 0 |
| **OR** | U1 | U2 | U1 OR U2 |
| (somme logique) | 1 | 1 | 1 |
| | 1 | 0 | 1 |
| | 0 | 1 | 1 |
| | 0 | 0 | 0 |
| **XOR** | U1 | U2 | U1 XOR U2 |
| (OU exclusif) | 1 | 1 | 0 |
| | 1 | 0 | 1 |
| | 0 | 1 | 1 |
| | 0 | 0 | 0 |
| **EQV** | U1 | U2 | U1 EQV U2 |
| | 1 | 1 | 1 |
| | 1 | 0 | 0 |
| | 0 | 1 | 0 |
| | 0 | 0 | 1 |
| **IMP** | U1 | U2 | U1 IMP U2 |
| (implication logique) | 1 | 1 | 1 |
| | 1 | 0 | 0 |
| | 0 | 1 | 1 |
| | 0 | 0 | 1 |

**Signes d'opération dans des chaînes**

Dans des chaînes on ne peut utiliser que le signe +, et ceci pour lier deux chaînes l'une à l'autre.
Exemple:

"AB"+"CD" devient "ABCD"

# G

# MESSAGES D'ERREURS

**Bad allocation table (60)**
La table d'allocation de la disquette est détruite.

**Bad drive name (62)**
Mauvais nom d'unité. Utiliser A:,B:,..

**Bad file mode (61)**
(mauvaise utilisation de fichier)
On utilise PUT, GET avec un fichier ouvert en séquentiel ou on ouvre un fichier dans un autre mode que Random, Output, Input ou Append.

**Bad file number (52)**
Mauvais numéro de fichier.

**Bad file name (56)**
(mauvais nom de fichier)
Le nom de fichier utilisé n'est pas normalisé (trop de caractères par exemple).

**Can't continue (17)**
(l'exécution ne peut se poursuivre)
On a tenté de poursuivre l'exécution d'un programme qui:
- a été stoppé à cause d'une erreur
- a été modifié après une interruption
- n'existe pas.

On peut cependant continuer par GOTO xx (RUN xx initialise les variables à 0)

**Direct statement in file (57)**
Commande directe dans un fichier.

**Disk full (66)**
(disque plein)
Tout l'espace disque est utilisé.

**Disk I/O error (69)**
(erreur disque)
Une erreur disque s'est produite pendant une lecture/écriture.
Le système d'exploitation ne peut rien faire.

**Disk Write protected (68)**
Le disque est protégé en écriture.

**Division by zero (11)**
(division par zéro)

**Field overflow (50)**
Le nombre d'octets attribués dans l'instruction FIELD est supérieur à 256.

**File already exists (65)**
(fichier déjà existant)
Le nom de fichier défini dans NAME existe déjà.

**File already open (54)**
(fichier déjà ouvert)
On essaie d'ouvrir en mode séquentiel OUTPUT un fichier déjà ouvert ou un KILL tente de supprimer un fichier ouvert.

**File not found (53)**
(fichier non trouvé)
Une instruction LOAD, KILL ou OPEN référence un fichier qui n'existe pas.

**File still open (64)**
Un fichier n'a pas été fermé par CLOSE.

**For without next (26)**
(FOR sans NEXT)
Un FOR sans NEXT a été détecté.

**Illegal direct (12)**
(illégal en mode direct)
L'instruction frappée n'est pas valide en mode direct, elle ne peut être exécutée que précédée d'un numéro de ligne.
Ex.: instruction INPUT-DEF FN

**Illegal function call (5)**
(appel illégal de fonction)
Un paramètre hors du domaine normal a été passé à une fonction arithmétique ou une fonction chaîne.
Ex.: – argument négatif pour SQR
– longueur spécifiée dans LEFT$,MID$,RIGHT$ non comprise entre 0 et 255.

**Input past end (55)**
(dépassement de fin de fichier)
Une instruction INPUT#.. est exécutée alors qu'il n'y a plus d'information à lire (fichier vide éventuellement). Pour éviter cette erreur, utiliser la détection de fin de fichier EOF.

**Internal error (51)**
(erreur interne)
Une erreur système s'est produite.

**Line buffer overflow (23)**
(dépassement du buffer de ligne)
On essaie d'entrer une ligne avec trop de caractères.

**Missing operand (24)**
(opérande manquant)
Une expression contient un opérateur sans opérande.

**NEXT without FOR (1)**
(NEXT sans FOR)
Une variable dans un NEXT ne correspond à aucun FOR.
Ex.: La ligne FOR correspondante a été effacée sans le NEXT associé.

**No resume (19)**
(pas d'instruction RESUME)
Une routine de traitement d'erreur qui ne contient pas d'instruction RESUME a été appelée.

**Out of data (4)**
(Data épuisées)
Un READ est exécuté alors qu'il n'y a plus de DATA à lire.
– On a oublié des DATA.
– On a oublié de programmer RESTORE

**Out of memory (7)**
(plus de mémoire centrale)
Il n'y a plus assez de place en mémoire centrale.
– On doit supprimer une partie du programme ou effacer des tableaux (ERASE).

**Out of string space (14)**
(plus de place pour les chaînes)
L'espace pour les chaînes n'est pas suffisant (voir CLEAR).

**Overflow (6)**
(dépassement de capacité)
Ex.: On a tenté de donner à une variable entière une valeur en dehors de −32768,+32767.

**Redimensionned array (10)**
(tableau redimensionné)
Un tableau est à nouveau dimensionné ou une dimension de tableau est déclarée pour un tableau non déclaré explicitement mais créé par BASIC (avec une dimension 10) parce qu'il a déjà été référencé (par une instruction A(4)=X par exemple).

**Resume without error (22)**
(RESUME sans erreur)
Une instruction RESUME a été exécutée alors qu'il n'y avait pas d'erreur.

**Sequential I/O only (58)**
Tentative de traiter un fichier séquentiel comme un fichier à accès aléatoire.

**String formula too complexe (16)**
(expression chaîne trop complexe)
Une expression du type chaîne est trop longue ou trop complexe. La décomposer en expressions plus courtes.

**String too long (15)**
(chaîne trop longue)
Une chaîne ne peut dépasser 255 caractères.

**Subscript out of range (9)**
(référence en dehors du domaine)
Un tableau est référencé en dehors de ses dimensions. Souvent, pour un tableau non déclaré qui a été dimensionné à 10 par BASIC parce que référencé (par une instruction A(4)=X par exemple).

**Syntax error (2)**
(erreur de syntaxe)
La ligne contient une erreur de syntaxe:
- parenthèses non appairées
- ponctuation incorrecte
- instruction n'existant pas
- etc.

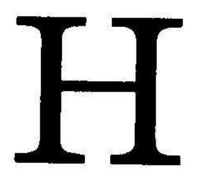
**Type mismatch (13)**
(désaccord entre numérique et chaîne)
Une valeur numérique est affectée à une chaîne où l'inverse.

**Too many files (67)**
Il y a plus de fichiers que le nombre prévu par MAXFILES.

**Undefined line (8)**
(numéro de ligne indéfini)
Une instruction référence une ligne qui n'existe pas.

**Undefined user fonction (18)**
(fonction utilisateur indéfinie)
Une fonction USR est appelée avant d'être définie.

**Unpritable error (23)**
(il n'y a pas de message pour cette erreur).



# SEQUENCES ESCAPE

Par 'séquences escape', on désigne un code spécial servant à effectuer certaines opérations à l'écran.

Pour introduire par exemple le code <ESC> B, indiqué comme séquence escape, entrez ce qui suit au clavier:

```
PRINT CHR$(27) + "B"
```

Le tableau suivant vous montre les diverses possibilités.

| | |
|---|---|
| <ESC> A | curseur à la ligne supérieure |
| <ESC> B | curseur à la ligne inférieure |
| <ESC> C | curseur d'une position vers la droite |
| <ESC> D | curseur d'une position vers la gauche |
| <ESC> H | curseur dans le coin supérieur gauche |
| <ESC> Y | <numéro de ligne + 32> <numéro de colonne + 32> placer le curseur à la position indiquée |
| <ESC> j | CLS |
| <ESC> E | CLS |
| <ESC> K | effacer jusqu'à la fin de la ligne |
| <ESC> J | effacer jusqu'à la fin de l'écran |
| <ESC> I | effacer toute la ligne |
| <ESC> L | insérer la ligne |
| <ESC> M | supprimer la ligne |
| <ESC> x4 | transformer la forme du curseur en un carré |
| <ESC> x5 | enlever le curseur |
| <ESC> y4 | transformer la forme du curseur en un trait |
| <ESC> y5 | réintroduire le curseur |

# I LES CODES CONTRÔLE

**Codes contrôle** Un certain nombre de touches ont une fonction spéciale lorsqu'elles sont combinées avec la touche CTRL. Quand on appuie sur la touche contrôle et en même temps le L, l'écran est effacé. Cette action est alors notée par CTRL/L. Ainsi CTRL/R veut dire: Appuyer sur R lorsque CTRL est enfoncée. L'aperçu suivant montre toutes les possibilités.

| | |
|---|---|
| CTRL/A | la touche suivante appelle un symbole de la série de symboles alternatifs. |
| CTRL/B | déplacez le curseur vers le mot précédent. |
| CTRL/C | arrêtez la commande AUTO. |
| CTRL/E | tous les symboles à partir du curseur sont effacés. |
| CTRL/F | déplacez le curseur vers le mot suivant. |
| CTRL/G | BEEP. |
| CTRL/H | correspond à la touche BS. |
| CTRL/I | correspond à TAB: vers l'arrêt suivant. |
| CTRL/J | déplacez le curseur à la ligne suivante. |
| CTRL/K | Déplace le curseur vers le coin supérieur gauche. |
| CTRL/L | correspond à CLS. |
| CTRL/M | correspond à la touche RETURN |
| CTRL/N | déplace le curseur vers la fin de la ligne (c'est-à-dire à la première position après le dernier symbole sur cette ligne). |
| CTRL/R | correspond à la touche INS. |
| CTRL/U | effacez la ligne. |
| CTRL/\ | correspond à la touche du curseur →. |
| CTRL/[ | correspond à la touche du curseur ←. |
| CTRL/∧ | correspond à la touche du curseur ↑. |
| CTRL/– | correspond à la touche du curseur ↓. |

# J

# STRUCTURE DES SYMBOLES

Cette annexe donne la liste de tous les symboles (caractères) et leurs codes correspondants.

Ceux-ci peuvent être utilisés par exemple dans la fonction CHR$. Ainsi le caractère A peut s'obtenir par PRINT 'A', mais aussi à l'aide de PRINT CHR$(65).

Dans le mode SCREEN 0 les deux colonnes de droite de la matrice avec lesquelles chaque symbole est défini, ne sont pas présentées.

On peut vérifier soi-même la différence en comparant:

```
10 SCREEN
20 PRINT CHR$(210)
```

et

```
10 SCREEN 1
20 PRINT CHR$(210)
```

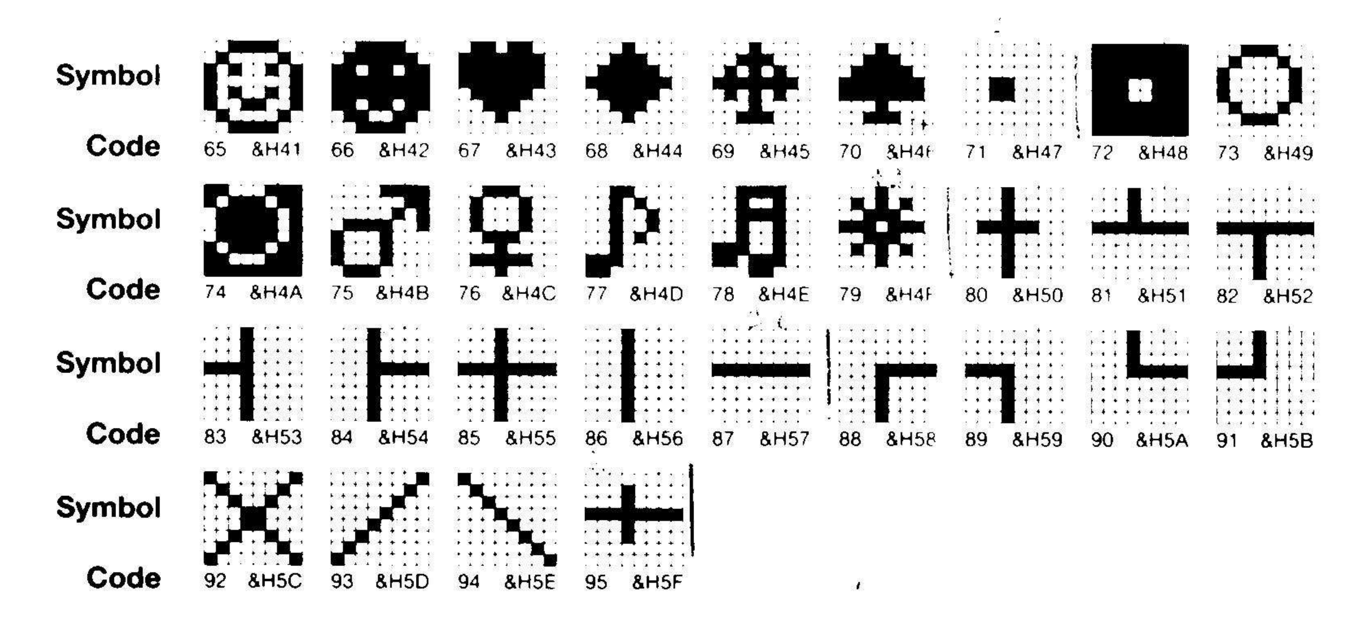
Les symboles standard sont reproduits dans les pages qui suivent.

**Symboles alternatifs**

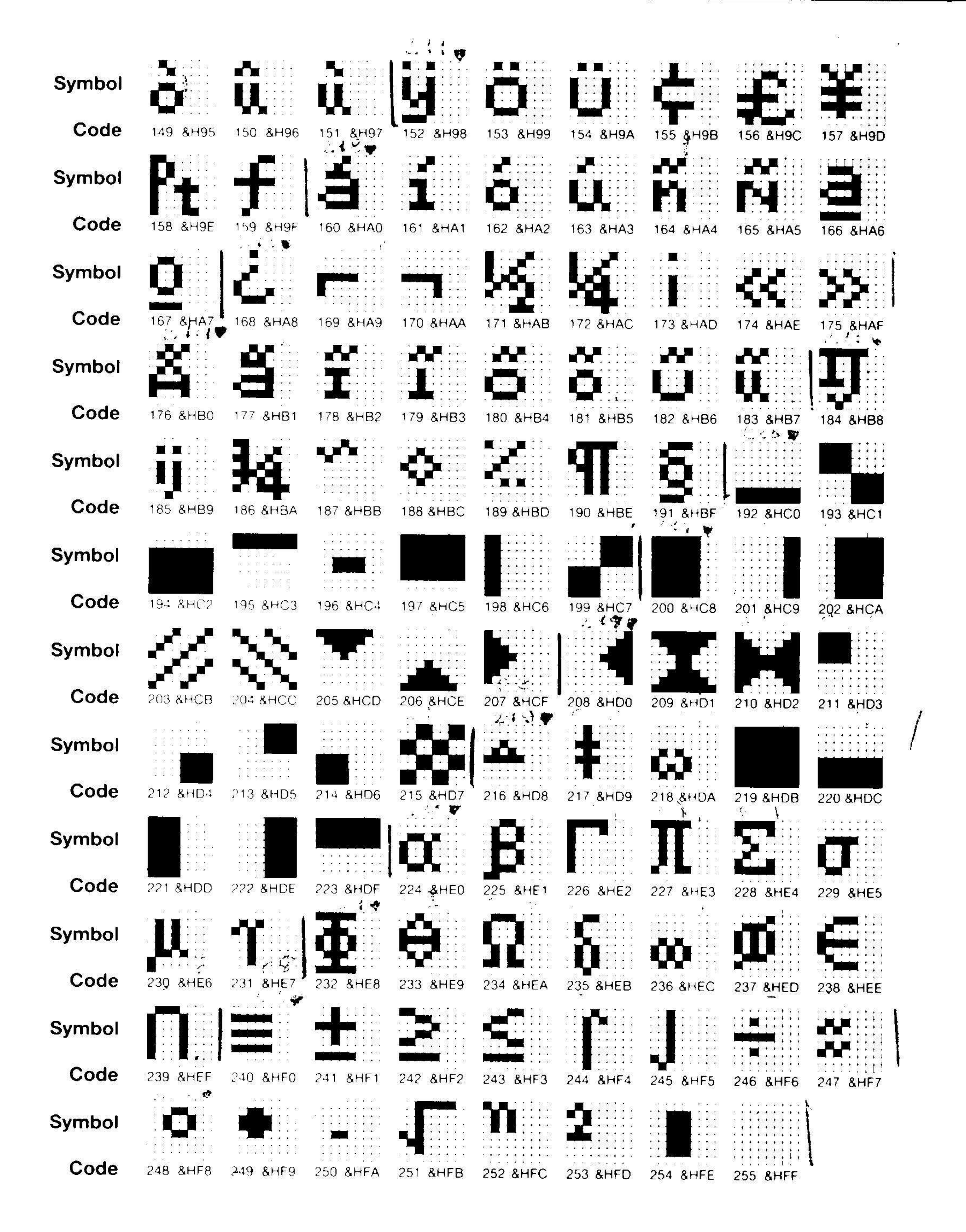
Ce sont les symboles qu'on obtient en plus des symboles déjà décrits, à savoir en incluant CHR$(1) dans l'instruction PRINT. Exemple:

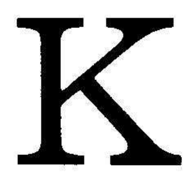
```
10 SCREEN 1
20 PRINT CHR$(65)
30 PRINT CHR$(1) + CHR$(65)
```

La ligne 20 génère un A et la ligne 30 a pour résultat qu'un visage est montré.



| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **Symbol** | | ! | " | # | $ | % | & | ' | ( |
| **Code** | 32 &H20 | 33 &H21 | 34 &H22 | 35 &H23 | 36 &H24 | 37 &H25 | 38 &H26 | 39 &H27 | 40 &H28 |
| **Symbol** | ) | * | + | , | - | . | / | Ø | 1 |
| **Code** | 41 &H29 | 42 &H2A | 43 &H2B | 44 &H2C | 45 &H2D | 46 &H2E | 47 &H2F | 48 &H30 | 49 &H31 |
| **Symbol** | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | : |
| **Code** | 50 &H32 | 51 &H33 | 52 &H34 | 53 &H35 | 54 &H36 | 55 &H37 | 56 &H38 | 57 &H39 | 58 &H3A |
| **Symbol** | ; | < | = | > | ? | @ | A | B | C |
| **Code** | 59 &H3B | 60 &H3C | 61 &H3D | 62 &H3E | 63 &H3F | 64 &H40 | 65 &H41 | 66 &H42 | 67 &H43 |
| **Symbol** | D | E | F | G | H | I | J | K | L |
| **Code** | 68 &H44 | 69 &H45 | 70 &H46 | 71 &H47 | 72 &H48 | 73 &H49 | 74 &H4A | 75 &H4B | 76 &H4C |
| **Symbol** | M | N | O | P | Q | R | S | T | U |
| **Code** | 77 &H4D | 78 &H4E | 79 &H4F | 80 &H50 | 81 &H51 | 82 &H52 | 83 &H53 | 84 &H54 | 85 &H55 |
| **Symbol** | V | W | X | Y | Z | [ | \ | ] | ^ |
| **Code** | 86 &H56 | 87 &H57 | 88 &H58 | 89 &H59 | 90 &H5A | 91 &H5B | 92 &H5C | 93 &H5D | 94 &H5E |
| **Symbol** | _ | ` | a | b | c | d | e | f | g |
| **Code** | 95 &H5F | 96 &H60 | 97 &H61 | 98 &H62 | 99 &H63 | 100 &H64 | 101 &H65 | 102 &H66 | 103 &H67 |
| **Symbol** | h | i | j | k | l | m | n | o | p |
| **Code** | 104 &H68 | 105 &H69 | 106 &H6A | 107 &H6B | 108 &H6C | 109 &H6D | 110 &H6E | 111 &H6F | 112 &H70 |
| **Symbol** | q | r | s | t | u | v | w | x | y |
| **Code** | 113 &H71 | 114 &H72 | 115 &H73 | 116 &H74 | 117 &H75 | 118 &H76 | 119 &H77 | 120 &H78 | 121 &H79 |
| **Symbol** | z | { | \| | } | ~ | | Ç | ü | é |
| **Code** | 122 &H7A | 123 &H7B | 124 &H7C | 125 &H7D | 126 &H7E | 127 &H7F | 128 &H80 | 129 &H81 | 130 &H82 |
| **Symbol** | â | ä | à | å | ç | ê | ë | è | ï |
| **Code** | 131 &H83 | 132 &H84 | 133 &H85 | 134 &H86 | 135 &H87 | 136 &H88 | 137 &H89 | 138 &H8A | 139 &H8B |
| **Symbol** | î | ì | Ä | Å | É | æ | Æ | ô | ö |
| **Code** | 140 &H8C | 141 &H8D | 142 &H8E | 143 &H8F | 144 &H90 | 145 &H91 | 146 &H92 | 147 &H93 | 148 &H94 |


Symbol
Code
149 &H95
150 &H96
151 &H97
152 &H98
153 &H99
154 &H9A
155 &H9B
156 &H9C
157 &H9D
Symbol
Code
158 &H9E
159 &H9F
160 &HA0
161 &HA1
162 &HA2
163 &HA3
164 &HA4
165 &HA5
166 &HA6
Symbol
Code
167 &HA7
168 &HA8
169 &HA9
170 &HAA
171 &HAB
172 &HAC
173 &HAD
174 &HAE
175 &HAF
Symbol
Code
176 &HB0
177 &HB1
178 &HB2
179 &HB3
180 &HB4
181 &HB5
182 &HB6
183 &HB7
184 &HB8
Symbol
Code
185 &HB9
186 &HBA
187 &HBB
188 &HBC
189 &HBD
190 &HBE
191 &HBF
192 &HC0
193 &HC1
Symbol
Code
194 &HC2
195 &HC3
196 &HC4
197 &HC5
198 &HC6
199 &HC7
200 &HC8
201 &HC9
202 &HCA
Symbol
Code
203 &HCB
204 &HCC
205 &HCD
206 &HCE
207 &HCF
208 &HD0
209 &HD1
210 &HD2
211 &HD3
Symbol
Code
212 &HD4
213 &HD5
214 &HD6
215 &HD7
216 &HD8
217 &HD9
218 &HDA
219 &HDB
220 &HDC
Symbol
Code
221 &HDD
222 &HDE
223 &HDF
224 &HE0
225 &HE1
226 &HE2
227 &HE3
228 &HE4
229 &HE5
Symbol
Code
230 &HE6
231 &HE7
232 &HE8
233 &HE9
234 &HEA
235 &HEB
236 &HEC
237 &HED
238 &HEE
Symbol
Code
239 &HEF
240 &HF0
241 &HF1
242 &HF2
243 &HF3
244 &HF4
245 &HF5
246 &HF6
247 &HF7
Symbol
Code
248 &HF8
249 &HF9
250 &HFA
251 &HFB
252 &HFC
253 &HFD
254 &HFE
255 &HFF



# MOTS RÉSERVÉS

Les mots suivants ne peuvent être utilises comme nom de variable ou comme début de nom de variables astérisques indiquent des extensions ultérieures.

| | | | | |
|---|---|---|---|---|
| ABS | DEFSTR | KEY | PAD | STRIG |
| AND | DELETE | KILL | PAINT | STRING$ |
| ASC | DIM | LEFT$ | PDL | SWAP |
| *ATTR$ | DRAW | LEN | PEEK | TAB( |
| ATN | DSKF | LET | PLAY | TAN |
| AUTO | DSKI$ | LFILES | POINT | THEN |
| BASE | DSKO$ | LINE | POKE | TIME |
| BEEP | ELSE | LIST | POS | TO |
| BIN$ | END | LLIST | PRESET | TROFF |
| BLOAD | EOF | LOAD | PRINT | TRON |
| BSAVE | EQV | LOC | PSET | USING |
| CALL | ERASE | LOCATE | PUT | USR |
| CDBL | ERL | LOF | READ | VAL |
| CHR$ | ERR | LOG | REM | VARPTR |
| CINT | ERROR | LPOS | RENUM | VDP |
| CIRCLE | EXP | LPRINT | RESTORE | VPEEK |
| CLEAR | FIELD | LSET | RESUME | VPOKE |
| CLOAD | FILES | *MAX | RETURN | WIDTH |
| CLOSE | FIX | MERGE | RIGHT$ | WAIT |
| CLS | FN | MID$ | RND | XOR |
| *CMD | FOR | MKD$ | RSET | |
| COLOR | *FPOS | MKI$ | RUN | |
| CONT | FRE | MKS$ | SAVE | |
| COPY | GET | MOD | SCREEN | |
| COS | GO TO | *MON | SET | |
| CSAVE | GOSUB | MOTOR | SGN | |
| CSNG | GOTO | NAME | SIN | |
| CSRLIN | HEX$ | NEW | SOUND | |
| CVD | IF | NEXT | SPACE$ | |
| CVI | IMP | NOT | SPC( | |
| CVS | INKEY$ | OCT$ | SPRITE | |
| DATA | INP | OFF | SQR | |
| DEF | INPUT | ON | STEP | |
| DEFDBL | INSTR | OPEN | STICK | |
| DEFINT | INT | OR | STOP | |
| DEFSNG | *IPL | OUT | STR$ | |

# APERÇU DES COMMANDES MSX-BASIC

Cet aperçu passe en revue toutes les instructions, commandes, fonctions et variables système en BASIC.

*Notations sur la syntaxe*
La syntaxe complète (c'est-à-dire la façon d'écrire) est donnée pour chaque instruction et chaque commande. A cette fin, les deux symboles suivants sont utilisés:

[ ... ] tout ce qui se trouve entre crochets est facultatif, c'est-à-dire peut être éventuellement omis;

< ... > les données entre ces crochets sont à remplir par l'utilisateur lui-même.
Cet aperçu utilise le terme 'fichier' et le terme 'file'.
Dans de nombreuses expressions on trouve trois points:

CALL <nom> [<chaîne>,<chaîne>...]

Les trois points à la fin de cette expression signifient 'l'expression peut être continuée de la façon indiquée'. Dans ce cas, on peut donc insérer davantage de chaînes séparées par une virgule.

Cet aperçu donne également la syntaxe complète de toutes les fonctions en BASIC et les variables système.
Deux notations sont utilisées:

**<X>** variable numérique ou expression que l'utilisateur doit remplir lui-même;

**<X$>** une variable chaîne ou expression que l'utilisateur doit remplir lui-même;

<dev> Dans une instruction, on utilise

| | |
|---|---|
| A à F | pour les unités de disque |
| CAS | pour le magnétophone |
| CRT | pour l'écran texte |
| GRP | pour l'écran graphique |
| LPT | pour l'imprimante |
| MEM | pour le mémoire disque |
| COM | pour l'interface RS232C |

<nom de fichier> 6 caractères au plus pour un fichier sur magnétophone à cassette.
8 au plus pour un fichier sur disquette.

La catégorie est mentionnée pour chaque tâche: instruction, commande, variable système ou function.
Après avoir exécuter une commande, le MSX-BASIC fait apparaître le message Ok sur l'écran et attend que la prochaine commande soit entrée au clavier.
Une fonction ou variable système ne peut être utilisée qu'avec une instruction ou une commande.

**ABS(<X>)**
Donne la valeur absolue de <X>.
Catégorie: fonction
Exemple: `PRINT ABS(-3)`

**ASC(<X$>)**
Donne le code ASCII du premier caractère de <X$>.
Catégorie: fonction
Exemple: `PRINT ASC("MSX")`

**ATN(<X>)**
Donne l'arctangente de <X>. (X en radians). Le résultat varie entre $-\pi/2$ et $\pi/2$.
Gatégorie: fonction
Exemple: `PRINT ATN(3)`

**AUTO[<numéros de ligne>][,[pas de progression>]]**
Génère automatiquement des numéros de ligne lors de la création d'un programme. Si un <numéro de ligne> n'est pas indiqué, l'ordinateur commence par le numéro de ligne 10, sinon par le <numéro de ligne> spécifié. Si le <pas de progression> n'est pas indiqué, le numéro de ligne est toujours incrémenté de 10, sinon le <pas de progression> est celui spécifié.
Quand un numéro de ligne est généré automatiquement mais a déjà été utilisé, on voit un astérisque * qui a valeur d'avertissement.
La commande est terminée par CTRL/C, c'est-à-dire que C est enfoncée en même temps que la touche CTRL.
Catégorie: commande
Exemple: `AUTO 100,50`

**BASE (<X>)**
Contient la première adresse du processeur de visualisation (tableaux VDP).
Le tableau suivant donne un aperçu des abréviations utilisées.

mt = mode de texte
mg = mode graphique
tp = tableau de patron
tps = tableau de patron de sprite
tas = tableau d'attribut de sprite
tc = tableau des codes de couleur
nt = nom du tableau

| <X> Signification | *<X> Signification* | <X> Signification | <X> Signification |
|---|---|---|---|
| 0 nt dans mt1 | 14 tps dans mg1 | 25 nt dans mg4 | 36 tc dans mg6 |
| 2 tp dans mt1 | 15 nt dans mg2 | 26 tc dans mg4 | 37 pt dans mg6 |
| 5 nt dans mt2 | 16 tc dans mg2 | 27 tp dans mg4 | 38 tas dans mg6 |
| 6 tc dans mt2 | 17 tp dans mg2 | 28 tas dans mg4 | 39 tps dans mg6 |
| 7 tp dans mt2 | 18 tas dans mg2 | 29 tps dans mg4 | 40 nt dans mg7 |
| 8 tas dans mt2 | 19 tps dans mg2 | 30 nt dans mg5 | 41 tc dans mg7 |
| 9 tps dans mt2 | 20 nt dans mg3 | 31 tc dans mg5 | 42 tp dans mg7 |
| 10 nt dans mg1 | 21 tc dans mg3 | 32 tp dans mg5 | 43 tas dans mg7 |
| 11 tc dans mg1 | 22 tp dans mg3 | 33 tas dans mg5 | 44 tps dans mg7 |
| 12 tp dans mg1 | 23 tas dans mg3 | 34 tps dans mg5 | |
| 13 tas dans mg1 | 24 tps dans mg3 | 35 nt dans mg6 | |

Dans l'adresse de BASE 0 à 19, on ne peut écrire qu'une valeur. Ceci implique que si l'on modifie une adresse de base dans le mode écran 2 ou 3, cela vaut également pour le mode écran 4 à 8.
Catégorie : variable système
Exemple :
```
10 SCREEN 0
20 PRINT BASE(2):END
```

**BEEP**
Génère un son bref.
On peut obtenir le même résultat avec PRINT CHR$(7).
Catégorie : instruction
Exemple :
```
10 FOR K=1 TO 1000:PRINT K:NEXT
20 BEEP:END
```

**BIN$(<X>)**
Convertit le nombre donné en une notation binaire.
<X> doit être compris entre - 32768 et 65535. Si <X> est négatif, on obtient la forme dénommée 'complément à deux'.
Catégorie : fonction
Exemple : `PRINT BIN$ (100)`

**BLOAD"<dev>:[<nom de fichier>]"[,R][,<déplacement>]**
Charge un programme écrit en code machine qui a été stocké avec une commande BSAVE.
Si l'on spécifie R, le programme est exécuté immédiatement après le chargement.
Si l'on spécifie un 'déplacement' (nombre entier), la nouvelle adresse de départ devient identique à l'ancienne adresse + le déplacement.
Catégorie: commande
Exemple: `BLOAD "CAS:TEST",R,&H20`

**BLOAD"<dev>:[<nom de fichier>]",S**
Comme la commande précédente, mais dans ce cas, S indique la mémoire vidéo RAM. Dans ce cas, <dev> ne peut être qu'une disquette (lettre de A à F).
Le fichier est toujours chargé dans la page active dans les modes écran 5,6,7 et 8. (voir SET PAGE)
Catégorie: commande
Exemple: `BLOAD "A:TST",S`

**BSAVE"<dev>:[<nom de fichier>]",<adresse de début, adresse de fin>[,<adresse de démarrage du programme>]**
Stocke un programme écrit en code machine sur un périphérique externe. Les adresses de début et de fin marquent le bloc de mémoire qui est copié sur le périphérique externe.
Catégorie: commande
Exemple: `BSAVE "CAS:TEST",&HC000,&HE0FF,&HC020`

**BSAVE"<dev>:[<nom de fichier>[",<adresse de début>,<adresse de fin>,S**
Comme ci-dessus, mais pour la mémoire VIDEO RAM. <dev> ne peut être qu'une disquette.
Le contenu de la page active (voir SET PAGE) est sauvegardé sur disquette dans les modes écran 5,6,7 et 8.
Catégorie: commande
Exemple: `BSAVE "A:TST",&H0,&HFFFF,S`

**CALL<nom>[(<chaîne>,<chaîne>...)]**
Commande générale pour exécuter certains sous-programmes stockés dans la cartouche ROM.
Les chaînes indiquées sont des constantes alphanumériques avec lequelles on peut transmettre des valeurs au sous-programme. Au lieu du terme CALL, on peut également utiliser le signe 'souligné' ( ).
Catégorie : instruction
Exemple : `_TALK`

**CALL COM ([<X>:] ,GOSUB<Y>)**
Cette instruction peut être utilisée s'il y a une interface RS232C. <X> indique le numéro de l'interface. La valeur par défaut est 0. <Y> indique vers quel sous-programme le saut doit être fait quand l'interface RS232C reçoit une interruption. Celle-ci doit préalablement être autorisée au moyen de l'instruction CALL COMON.
Catégorie : instruction
Exemple :

```
 10 OPEN "COM:" FOR INPUT AS#1
 20 CALL COMON ("")
 30 CALL COM(,GOSUB 60)
 40 PRINT "PROGRAMME POUR RECEVOIR DES DONNEES"
 50 GOTO 50
 60 PRINT "JE RECOIS MAINTENANT:";
 70 A$=INPUT$(1,#1)
 80 PRINT A$
 90 RETURN
100 END
```

**CALL COMBREAK(["<n:>"],<code de spécification>)**
Permet de transmettre des caractères d'interruption par l'interface RS232C. Le <code de spécification> indiqué est alors utilisé comme ordre pour interrompre la communication (nombre entre 3-32767). Cette commande ne peut être active que si la RS232C est déjà ouverte.
Par <n>, on désigne l'interface RS232C concernée. La valeur par défaut est 0.
Catégorie: instruction
Exemple:
```
10 OPEN "COM:"FOR OUTPUT AS#1
20 CALL COMBREAK (,3)
30 CLOSE
40 END
```

**CALL COMDTR(["<n:>"],<expression>)**
Indique que l'interface RS232C est temporairement hors service, si l'expression est égale à 0. Cette commande est active seulement si la RS232C est déjà ouverte.
Par <n>, on désigne l'interface RS232C concernée. La valeur par défaut est 0.
Catégorie: instruction
Exemple:
```
10 OPEN "COM:"FOR OUTPUT AS#1
20 CALL COMDTR (,0)
30 CLOSE
40 END
```

**CALL COMINI[([<expression chaîne>][,[<Rx débit en bauds>][,[<Tx débit en bauds>][,[<limite horaire>]]]])]**
Instruction pour initialiser l'interface RS232C.
Signification des champs:

<Rx débit en bauds>: débit en bauds (bits/sec) à la réception.
Par défaut 1200 bauds:

<Tx débit en bauds>: débit en bauds (bits/sec) à l'emission.
Par défaut 1200 bauds:

<limite horaire>: temps (en sec) que l'ordinateur attendra avant de signaler une erreur.
Limite horaire 0 signifie que l'ordinateur attendra toujours.
Par défaut 0:

<expression chaîne>:
celle-ci a la forme:
"[<knl>:][<bl>[<pa >[<sl>[<x>[<hs>[<ilf>[<slf>[<ss>]]]]]]]]"

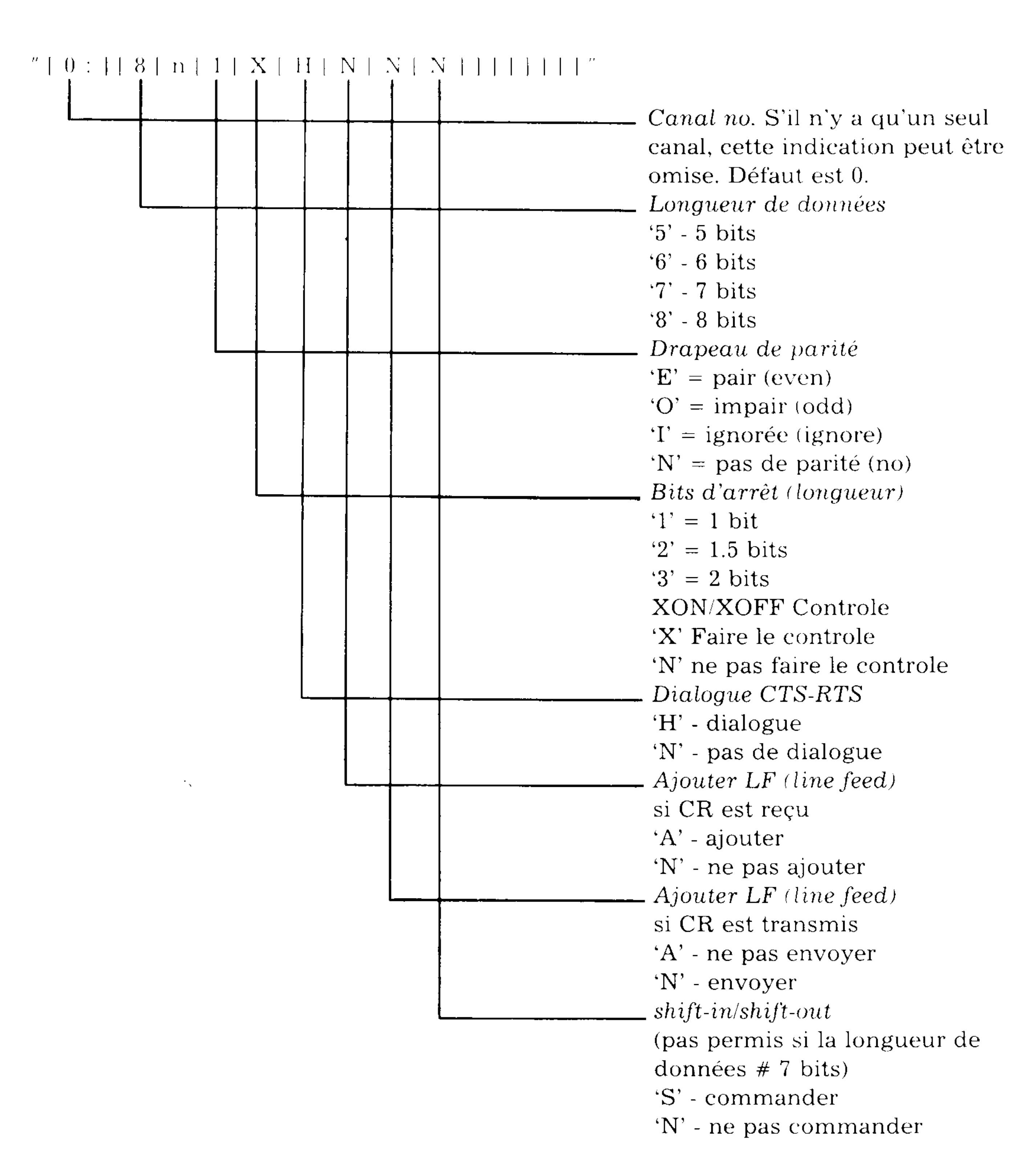
<knl>: numéro de canal: numéro de l'interface utilisée.
Nécessaire seulement quand il y a plus d'une interface.
Par défaut: '0':

<bl>: longueur d'octet: longueur (en bits) de l'unité de données (octet) transmise en une seule fois. Obligatoirement '5', '6', '7' ou '8'.
Par défaut: '8'

<pa>: parité
'E' = parité paire (even)
'O' = parité impaire (odd)
'I' = parité ignorée (ignore)

'N' = pas de parité (no)
Par défaut: 'E'

<sl>: longueur d'arrêt: longueur (en bits) de la pause entre la transmission de deux octets:
'1' = 1 octet
'2' = 1.5 octet
'3' = 2 octets
Par défaut:'1'



<x>: XON/XOFF commande
'X' = commander
'N' = ne pas commander
Par défaut: 'X'

<hs>: dialogue CTS-RTS
'H' = dialogue
'N' = pas de dialogue
Par défaut: 'H'

<ilf>: insertion d'un signe d'avancement de ligne (LF) chaque fois qu'un signe retour chariot est reçu (CR).
'A' = insérér
'N' = ne pas insérer
Par défaut: 'N'

<slf>: transmettre un signe avancement de ligne (LF) après chaque retour chariot (CR).
'A' = ne pas transmettre d'avancement de ligne
'N' = transmettre avancement de ligne
Par défaut: 'N'

<ss>: commande shift-in/shift-out
'S' = commander
'N' = ne pas commander
Par défaut: 'N'

Catégorie: instruction
Exemple: `CALL COMINI ("0:7N2XHAAN,75,1200)`

**CALL COMON("[<n:>]")**
**CALL COMOFF("[<n:>]")**
**CALL COMSTOP("[<n:>]")**
Par <n>, on désigne l'interface RS232C.
En utilisant un certain nombre de commandes CALL, vous pouvez laisser ou non un programme BASIC exécuter automatiquement un sous-programme dès qu'un caractère est reçu via l'interface RS232C. Le fonctionnement de ces commandes est parfaitement identique à celui de ON <événement> GOSUB <numéro de ligne>
Catégorie: instruction
Exemple: voir CALL COM

**CALL COMSTAT(["<n:>"],<nom de variable>)**
Par <n>, on désigne l'interface RS232C concernée.
Si un message Device I/O-error s'affiche lors d'une communication au moyen d'une interface RS232C, on donne à la variable une valeur qui permet d'expliquer l'anomalie éventuelle.
La table suivante donne les possibilités. Dans un certain nombre de cas les termes standard anglais ont été repris:

| *Bit no.* | *Signification* |
|---|---|
| 15 | Dépassement de capacité du buffer<br>0 - pas de depassement<br>1 - dépassement |
| 14 | Temps d'attente dépassé<br>0 - pas d'erreur<br>1 - erreur |
| 13 | Erreur de Transmission<br>0 - pas d'erreur<br>1 - erreur |
| 12 | Erreur de fonctionnement<br>0 - pas d'erreur<br>1 - erreur |
| 11 | Erreur de parité<br>0 - pas d'erreur<br>1 - erreur |
| 10 | Touche Control-STOP enfoncée<br>0 - pas enfoncée<br>1 - enfoncée |
| 9 | pas utilisé |
| 8 | pas utilisé |
| 7 | Clear to Send (prêt à transmettre)<br>0 - pas prêt<br>1 - prêt |
| 6 | Timer/compteur<br>0 - timer/compteur mis hors service<br>1 - timer/compteur est attribué |
| 5 | pas utilisé |
| 4 | pas utilisé |
| 3 | Données prêtes<br>0 - pas prêtes<br>1 - prêtes |
| 2 | Détection d'arrêt<br>0 - non<br>1 - oui |
| 1 | Indicateur sonore<br>0 - non<br>1 - oui |
| 0 | Detection de porteuse<br>0 - non<br>1 - oui |

Catégorie: instruction

Exemple:

```
10 OPEN "COM:" FOR INPUT AS#1
20 CALL COMSTAT(,A%)
30 A$="0000000000000000"+BIN$(A%)
40 PRINT RIGHT$(A$,16)
50 CLOSE
60 END
```

**CALL COMTERM ["<n>"]**
Démarre le mode terminal de l'interface RS232C. Le signe <n> indique l'interface RS232C concernée; la valeur par défaut est 0. Le canal RS232C doit d'abord être fermé avec CLOSE avant que CALL COMTERM puisse être exécuté.

Les touches F6, F7 et F8 ont une signification spéciale:
**F6** active ou désactive le 'mode littéral'. Dans cet état, les codes des caractères de contrôle sont transposés à l'écran en leur ajoutant 64. (1 + 64 = 65 donne A à l'écran).
**F7** mode duplex half/full. Dans le 'mode half duplex', les caractères entrés au clavier s'affichent à l'écran tout en étant envoyés par le canal RS232C.
**F8** active ou désactive l'imprimante.
Catégorie: commande
Exemple: `CALL COMTERM`

**CALL FORMAT ou _FORMAT**
Pour formater un disque non encore utilisé.
Attention! Un disque déjà utilisé sera effacé complètement par la commande FORMAT!
Catégorie: commande
Exemple: `CALL FORMAT`

**CALL MEMINI [(<X>)]**
Réserve une partie de la mémoire pour l'utilisation du disque mémoire. Si on indique une valeur, l'espace mémoire réservé s'obtient au moyen de la formule suivante:

(INT ((<X>–1023)/256+1)*256
X doit être supérieur à 1023.

CALL MEMINI doit toujours être utilisé avant d'utiliser une partie de la mémoire comme disque mémoire. Pour désactiver la fonction de disque mémoire, on donne la commande CALL MEMINI (0).
Le nom du périphérique est 'MEM:'
Catégorie: instruction
Exemple:
```
CALL MEMINI
SAVE "MEM:PROG.BAS"
```

**CALL MFILES**
Affiche le répertoire de tous les fichiers stockés dans le disque mémoire. Si le disque mémoire n'a pas été initialisé au préalable au moyen de la commande CALL MEMINI, la commande CALL MFILES entraînera l'affichage du message d'erreur 'disk offline'.
Catégorie: commande
Exemple: `CALL MFILES`

**CALL MKILL ("<nom de fichier>")**
Cette commande permet d'effacer un fichier stocké dans le disque mémoire.
Catégorie: commande
Exemple: `CALL MKILL("TEST")`

**CALL MNAME ("<nom de fichier1>" AS "<nom de fichier2>")**
Cette commande permet de changer le nom d'un fichier se trouvant dans le disque mémoire.

Exemple: CALL MNAME('PROG.BAS' AS 'PR1.BAS').
Le fichier appelé PROG.BAS sera désormais appelé PR1.BAS. Si le disque mémoire n'a pas été initialisé au préalable au moyen de la commande CALL MEMINI, la commande CALL MNAME entraînera l'affichage du message d'erreur 'disk offline'.
Catégorie: commande
Exemple: `CALL MNAME("PROG.BAS" AS "PR1.BAS").`

**CALL SYSTEM**
Commande pour quitter MSX-BASIC et remettre le système sous le contrôle du système d'exploitation MSX-DOS. A condition que le MSX-DOS ait été chargé au moment de l'initialisation.
Catégorie: commande
Exemple: `CALL SYSTEM`

**CDBL(<X>)**
Convertit <X> en un nombre en double précision.
Catégorie: fonction
Exemple: `PRINT CDBL(7/6)`

**CHR$ (<code ASCII>)**
Donne le caractère correspondant au <code ASCII>
Catégorie: fonction
Exemple: `PRINT CHR$(65)`

**CINT(<X>)**
Convertit X en un entier avec arrondi. La valeur doit se situer entre −32768 et 32767.
Catégorie: fonction
Exemple: `PRINT CINT(7/6)`

**CIRCLE[STEP](<x,y>),<r>[,[<couleur>][,[<angle de début>]
[,[<angle de fin>][,<aplatissement>]]]]**
Génère une ellipse dont <x,y> indique le centre, r indique le rayon. Eventuellement, on peut en dessiner seulement une partie, en spécifiant un angle de début et un angle de fin (indications en radians). Si l'on omet STEP, le système de coordonnées normal est utilisé. Avec STEP, x et y sont estimés par rapport au dernier point place sur l'écran.
<applatissement> est un nombre qui indique le rapport entre le rayon vertical et le rayon horizontal.
Catégorie: instruction
Exemple:

```
10 SCREEN 2
20 CIRCLE (127,95),50,,,1.4
30 GOTO 30
```

**CLEAR[<espace mémoire pour chaines>[,<adresse la plus élevée>]]**
Initialise toutes les variables, tous les tableaux, toutes les fonctions définiés par DEFFN, et réserve un espace mémoire pour le stockage des caractères (chaîne, lettres). Enfin CLEAR ferme tous les fichiers éventuellement ouverts.
L'adresse la plus élevée détermine l'adresse la plus haute pour l'utilisation en BASIC. Sans CLEAR, l'ordinateur ne réserve que 200 octets pour le stockage de lettres etc.
Catégorie: instruction
Exemple: `10 A-10:B$ "TEST"`

```
20 PRINT A,B$
30 CLEAR
40 PRINT A,B$:END
```

**CLOAD["<nom de fichier>"]**
Cette commande charge à partir d'une cassette, un programme, sous un nom (6 lettres et/ou chiffres au plus). Ainsi, CLOAD 'PROG4' a pour résultat que le programme répondant au nom PROG4, est chargé à partir.de la cassette. Si quelque chose ne va pas, il faut interrompre le chargement par CTRL/STOP, rembobiner la bande, et recommencer le chargement.
Quand on omet le nom de fichier, le premier programme trouvé sur la cassette est chargé.
Catégorie: commande
Exemple: `CLOAD"TEST"`

**CLOAD?["<nom de fichier>"]**
Commande permettant de comparer un programme sur cassette au programme en mémoire. Si tout est en règle, 'Ok' s'affiche. S'il y a une erreur, 'Verify error' apparaît à l'écran.
Catégorie: commande
Exemple: `CLOAD?"TEST"`

**CLOSE[#][<numéro de fichier>][,[#]<numéro de fichier>...]**
Ferme les fichiers portant les numéros spécifiés. Quand on ne spécifie pas de numéro, tous les fichiers sont fermés.
Catégorie: instruction
Exemple:
```
10 MAXFILES=1
20 OPEN "CAS:TEST" FOR OUTPUT AS#1
30 A$="MSX"
40 PRINT#1,A$
50 CLOSE#1
60 END
```

**CLS**
Efface l'écran.
Catégorie: instruction
Exemple:
```
10 CLS
20 END
```

**COLOR [<couleur de l'écriture>][,<couleur du fond>][,<couleur du pourtour>]**
Cette instruction permet de déterminer la couleur de la partie indiquée. La couleur qui apparaît est la couleur standard ou la couleur donnée par COLOR =-instruction (exception : mode écran 8). Le tableau suivant reproduit les couleurs standard et la manière de les obtenir en mélangeant le rouge, le vert et le bleu.

| *No du code couleur* | *Nom* | *Intensité* | | |
|---|---|---|---|---|
| | | *Rouge* | *vert* | *bleu* |
| 0 | Transparent | 0 | 0 | 0 |
| 1 | Noir | 0 | 0 | 0 |
| 2 | Vert | 1 | 6 | 1 |
| 3 | Vert clair | 3 | 7 | 3 |
| 4 | Bleu foncé | 1 | 1 | 7 |

| *No du code couleur* | *Nom* | | *Intensité* | |
|---|---|---|---|---|
| | | *Rouge* | *vert* | *bleu* |
| 5 | Bleu clair | 2 | 3 | 7 |
| 6 | Rouge foncé | 5 | 1 | 1 |
| 7 | Bleu ciel | 2 | 6 | 7 |
| 8 | Rouge | 7 | 1 | 1 |
| 9 | Rouge clair | 7 | 3 | 3 |
| 10 | Jaune foncé | 6 | 6 | 1 |
| 11 | Jaune clair | 6 | 6 | 4 |
| 12 | Vert foncé | 1 | 4 | 1 |
| 13 | Magenta | 6 | 2 | 5 |
| 14 | Gris | 5 | 5 | 5 |
| 15 | Blanc | 7 | 7 | 7 |

Mode écran 6
Dans ce mode, on peut obtenir un pourtour avec une structure hachurée. Lorsque la <couleur du pourtour> est un chiffre entre 0 et 3, la couleur correspond au numéro de code. Pour les valeurs de 16 à 31, on obtient une structure hachurée selon les combinaisons suivantes:

| *<couleur du pourtour>* | *Combinaison des codes* |
|---|---|
| 16 | 0 et 0 |
| 17 | 0 et 1 |
| 18 | 0 et 2 |
| 19 | 0 et 3 |
| 20 | 1 et 0 |
| 21 | 1 et 1 |
| 22 | 1 et 2 |
| 23 | 1 et 3 |
| 24 | 2 et 0 |
| 25 | 2 et 1 |
| 26 | 2 et 2 |
| 27 | 2 et 3 |
| 28 | 3 et 0 |
| 29 | 3 et 1 |
| 30 | 3 et 2 |
| 31 | 3 et 3 |

Mode écran 8
Dans ce mode, le code couleur se trouve compris entre 0 et 255. La couleur est déterminée au moyen de la formule : code couleur=4*R+32*V+B.
R, V et B déterminent l'intensité du rouge, du vert et du bleu. Pour le rouge et le vert, l'intensité doit être comprise entre 0 et 7. Le bleu doit être compris entre 0 et 3.
De la sorte, une couleur avec une intensité de rouge, vert et bleu de 7, 5 et 2 respectivement donnera le code couleur 190 (=4*7+32*5+2).
Catégorie: instruction
Exemple: `COLOR 5,5,7`

**COLOR=(<X>,<XX>,<YY>,<ZZ>)**

Cette instruction permet de donner au code couleur X une répartition différente de l'intensité (voir COLOR) par rapport au mélange standard. Les valeurs XX, YY et ZZ représentent l'intensité (entre 0 et 7) du rouge, du vert et du bleu respectivement. Le code couleur 0 est généralement attribué au "transparent" : cette couleur n'est donc pas visible.

Catégorie: instruction

Exemple: `COLOR=(9,3,7,2)`

**COLOR [=NEW]**

Cette instruction permet d'attribuer à nouveau aux codes couleur la répartition standard de l'intensité (voir COLOR).

Catégorie: instruction

Exemple: `COLOR=NEW`

**COLOR=RESTORE**

Cette instruction donne aux codes couleur la répartition de l'intensité chargée dans la mémoire vidéo par l'instruction BLOAD...,S. Le tableau avec la répartition de l'intensité est stocké de la manière suivante dans la mémoire vidéo:

| *Mode écran* | *Adresse de la mémoire vidéo* |
|---|---|
| Mode écran 0 (40 car.) | &H0400 – &H0420 |
| Mode écran 0 (80 car.) | &H0F00 – &H0F20 |
| Mode écran 1 | &H2020 – &H2040 |
| Mode écran 2 | &H2020 – &H2040 |
| Mode écran 3 | &H2020 – &H2040 |
| Mode écran 4 | &H2020 – &H2040 |
| Mode écran 5 | &H7680 – &H76A0 |
| Mode écran 6 | &H7680 – &H76A0 |
| Mode écran 7 | &HFA80 – &HFAA0 |
| Mode écran 8 | &HFA80 – &HFAA0 |

Dans les modes écran 5, 6, 7 et 8, on peut déterminer plus d'une page (voir SET PAGE). Les adresses qui contiennent à présent les tableaux des codes couleur doivent être déterminées par les calculs suivant.

| *Mode écran* | *Calcul* |
|---|---|
| Mode écran 5 | No de page x &H08000 + &H7680 |
| Mode écran 6 | No de page x &H08000 + &H7680 |
| Mode écran 7 | No de page x &H10000 + &HFA80 |
| Mode écran 8 | No de page x &H10000 + &HFA80 |

Catégorie: instruction

Exemple:
```
10 SCREEN 0:WIDTH 40:COLOR 15
20 COLOR=(15,0,7,0)
30 PRINT "XYZ"
40 BSAVE "COL1.PIC",&H400,&H420,S
50 COLOR=NEW
60 FOR I=0 TO 2000:NEXT
70 BLOAD "COL1.PIC",S
80 COLOR=RESTORE
90 END
```

**COLOR SPRITE (<X>)=<Y>**

Cette instruction sert à déterminer la couleur du sprite indiqué (uniquement dans les modes écran 4, 5, 6, 7 et 8). X est le numéro du sprite tel qu'il est déterminé par SPRITE$(X). Lorsque Y est compris entre 0 et 15, il s'agit d'un code couleur. Si nous ajoutons 32 à ce nombre, l'instruction ON SPRITE GOSUB n'entraînera plus de collision avec un autre sprite. En ajoutant 64, la fonction suivante est appelée : refuser la priorité et la survenance d'une collision entre les sprites et manipuler une opération OR logique pour déterminer la couleur. Dans ce cas aussi, si une collision survient avec un autre sprite, il n'y aura pas de saut vers un autre sous-programme.

Catégorie: instruction

Exemple:
```
10 SCREEN 5,0
20 B$=""
30 FOR I=1 TO 8:READ A:B$=B$+CHR$(A):NEXT
40 SPRITE$(0)=B$
50 COLOR SPRITE(0)=12
60 FOR I=0 TO 212:PUT SPRITE 0,(I,I),,0:NEXT
70 DATA 24.60,126,255.36,36,66,129
80 END
```

**COLOR SPRITE$(<X>)=<X$>**

Cette instruction vous permet d'attribuer une couleur à une partie de lutin (uniquement dans les modes écran 4, 5, 6, 7 et 8). X indique la couleur et X$ est une chaîne qui compte de 1 à 16 caractères.

Chaque caractère correspond à une ligne horizontale du sprite. Lorsque le code ASCII de X$ est compris entre 0 et 15, il s'agit d'un code couleur. En ajoutant 32, il se produit une situation dans laquelle l'ordinateur ne réagit pas aux collisions lorsque cette partie du sprite coïncide avec un autre sprite (voir instruction précédente). En ajoutant 64, l'ordinateur ne réagit pas aux collisions et la couleur est déterminée par une opération OR logique (voir aussi l'instruction précédente). En ajoutant 128, une 'ligne de sprite' est déplacée de 32 points vers la gauche.

Catégorie: instruction

Exemple:
```
 10 SCREEN 5,0
 20 B$=""
 30 FOR I=1 TO 8:READ A:B$=B$+CHR$(A):NEXT
 40 SPRITE$(0)=B$
 50 PUT SPRITE 0,(100,100),,0
 60 FOR I=0 TO 2000:NEXT
 70 COLOR SPRITE$(0)=CHR$(12)+CHR$(1)+CHR$(130)
 80 FOR I=0 TO 2000:NEXT
 90 DATA 24,60,126,255,36,36,66,129
100 END
```

**CONT**

Avec cette commande, l'exécution du programme est reprise, et ceci, à partir du moment où il a été interrompu par CTRL/STOP, ou par les instructions STOP ou END.

Catégorie: commande

Exemple: `CONT`

**COPY (<X1>,<Y1>) - (<X2>,<Y2>)[,<XX> TO (<X3>,<Y3>)[,<YY>[,<opération>]]**
**COPY (<X1>,<Y1>) - (<X2>,<Y2>)[,<XX>] TO <Tableau>**
**COPY (<X1>,<Y1>) - (<X2>,<Y2>)[,<XX>] TO "[<dev>:]<nom de fichier>"**
**COPY <Tableau>[,direction] TO (<X3>,<Y3>)[,<YY>[,<opération>]]**
**COPY "[<dev:>]<nom de fichier>"[direction] TO (<X3>,<Y3>)[,<YY>[,<opération>]]**
**COPY<Tableau> TO "[<dev:>]<nom de fichier>"**
**COPY"[<dev:>]<nom de fichier>" TO <Tableau>**
**COPY"[<dev:>]<nom de fichier 1>" TO "[<dev:>]<nom de fichier 2>"**

Cette instruction permet de copier une partie de l'écran graphique dans un tableau ou un fichier (uniquement dans les modes écran 5, 6, 7 et 8). X1 et Y1 déterminent les coordonnées du point de départ de la partie de l'écran que l'on veut copier. X2 et Y2 déterminent les coordonnées du point final de la partie à copier.
X3 et Y3 déterminent le point de départ de l'endroit où la copie doit être reproduite. Dans les modes écran 5, 6 et 7 et 8, plus d'une page peut être utilisée (voir SET PAGE). XX détermine la page 'source' et YY la page 'de destination'. Lorsque XX et YY ne sont pas indiquées, il s'agit de la page active, c'est-à-dire la page dans laquelle les résultats des opérations à l'écran sont visibles.
<direction> détermine la rotation de l'image:

| *<direction>* | *<Rotation de l'image>* |
|---|---|
| 0 Pas de retournement | |
| 1 Retournement | gauche-droite |
| 2 Retournement | haut-bas |
| 3 Retournement | gauche-droite et haut-bas |

<Tableau> est le nom d'un tableau dont la dimension est déterminée par la formule suivante:

```
INT((<PIXEL>*(ABS(X2-X1)+1*(ABS(Y2-Y1)+1)+7)/8)+4
```

On adopte les conventions suivantes:
<pixel> = 4 dans les modes écran 5, 7 et 8 et <pixel> = 2 dans le mode écran 6.
<opération> est un opérateur logique. Vous trouverez une explication détaillée à ce sujet à l'instruction PSET.
<dev:> ne peut concerner que le lecteur de disquettes (A à F).
Catégorie: instruction
Exemple:

```
 10 SCREEN 6
 20 X1=50:Y1=50:X2=100:Y2=0
 30 CIRCLE (X1,Y1),40,3
 40 PAINT (X1,Y1),3
 50 AA=INT((4*(ABS(X2-X1)+1)*(ABS(Y2-Y1)+1)+7)/8)+4
 60 DIM A(AA)
 70 COPY (X1,Y1)-(X2,Y2) TO A
 80 FOR I=1 TO 2000:NEXT
 90 COPY A,2 TO (150,150)
100 FOR I=1 TO 2000:NEXT
110 END
```

**COPY SCREEN [<X>]**

Cette instruction sert à indiquer qu'un signal vidéo donné doit être converti en données numériques. X en détermine la procédure.
**0** numérise le signal et le place dans la page active (page qui est affichée).
**1** numérise 2 trames et place la première dans la page active et la seconde dans la page dont le numéro est inférieur de un à celui de la page active (voir instruction SET PAGE).
La valeur par défaut 0.
Catégorie: instruction
Exemple: `COPY SCREEN1`
*Attention:* cette instruction ne fonctionne que si l'ordinateur est doté d'un convertisseur numérique vidéo.

**COS (<X>)**

Détermine le cosinus de X.
Catégorie: fonction
Exemple: `PRINT COS(3.1415/6)`

**CSAVE"<nom de fichier>"[,<débit en bauds>]**

Commande permettant de sauvegarder un programme sur cassette (voir aussi CLOAD).
Le débit, en bauds est
**1** 1200 bauds, ou
**2** 2400 bauds.
Si <débit en bauds> est omis, un débit de 1200 bauds est utilisé.
Le débit en bauds peut être réglé également au moyen de SCREEN.
Catégorie: commande
Exemple: `CSAVE"TEST"`

**CSGN(<X>)**

Convertit la valeur de X en un nombre en simple précision.
Catégorie: fonction
Exemple: `PRINT CSNG(9/7)`

**CSRLIN**

Donne la coordonnée y du curseur.
Catégorie: fonction
Exemple:

```
10 SCREEN 0:LOCATE 10,20:PRINT CSRLIN
20 END
```

**CVI (<chaîne à 2 octets>)**
**CVS (<chaîne à 4 octets>)**
**CVD (<chaîne à 8 octets>)**

Ces fonctions décodent les chaînes transmises en entier (CVI), simple précision (CVS) ou double précision (CVD). Les chaînes doivent être codées selon les fonctions inversées (MKI$, MKS$ et MKD$, voir sous ces fonctions), ou doivent être lues à partir d'un fichier à accès aléatoire. On peut utiliser ces fonctions pour retrouver des nombres stockés dans un fichier à accès aléatoire. Dans une commande FIELD, on ne peut indiquer que des variables chaîne; ces fonctions permettent de convertir des nombres codés sous forme de chaîne en nombres arithmétiques.
Catégorie: fonction
Exemple: `PRINT CVD(D$):CVS(S$):CVI(I$)`

**DATA <n1>[,<n2>...]**

Cette instruction permet d'insérer une liste de valeurs déterminées dans un programme, celles-ci pouvant alors être lues une par une au moyen de l'instruction READ. La liste peut se composer de valeurs numériques et de chaînes (entre guillemets (")), et celles-ci doivent être séparées par des virgules. READ récupère les valeurs de manière séquentielle. On peut relire une liste DATA depuis le début, en exécutant d'abord l'instruction RESTORE.

Catégorie: instruction

Exemple:
```
10 READ A,B,C:PRINT A,B,C
20 DATA 5,6,7
30 END
```

**DEF FN <nom de fonction>[(<listes d'arguments>)]= <définition de fonction>**

Permet de définir ses propres fonctions. Le nom sous lequel la fonction doit être appelée est le <nom de fonction> précédé de FN. Dans les fonctions de chaîne, le nom finit par $. Eventuellement, on peut inclure dans la fonction des arguments qui sont utilisés alors dans la <définition de fonction>. Les noms des arguments servent seulement à définir la fonction et n'ont pas d'influence sur les variables du programme principal de même nom. Les arguments doivent être séparés par une virgule.

Dans la <définition de fonction>, peuvent également figurer des noms de variables qui n'existent pas dans la <liste d'arguments>, mais qui, pour l'appel de la fonction, ont déjà reçu une valeur dans le programme principal. La <définition de fonction> ne doit pas dépasser une ligne informatique.

Exemple:
```
10 DEF FNAB(X,Y)=X^2+Y*Z
20 Z=5
30 I=2:J=3
40 T=FNAB(I,J)
50 PRINT T
60 END
```

**DEF<type><indication de lettre>**

Cette instruction garantit que toutes les variables commençant par les <lettre(s)> indiquée(s) sont du <type> spécifié. Les <types> possibles sont:

**INT** entier
**SGN** simple précision
**DBL** double précision
**STR** chaîne

Un signe de déclaration de type (donc %, #,$) a la priorité sur l'instruction DEF.

Exemples:

| | |
|---|---|
| `10 DEFDBL A-E` | Toutes les variables commençant par A, B, C, D ou E sont des variables en double précision. |
| `10 DEFSTR A` | Toutes les variables commençant par la lettre A sont des expressions chaîne. |

Catégorie: instruction

Exemple:
```
10 DEFINT I
20 I=3/2:PRINT I
30 END
```

**DEF USR[<nombre>]=<adresse de mémoire>**
Ainsi, on précise l'adresse de début d'un sous-programme en langage machine. Le <nombre> doit être compris entre 0 et 9 et est attribué comme nom au programme en langage machine. Si le <nombre> est omis, 0 est pris par défaut. A partir de ce <nombre> et au moyen de la fonction USR, le sous-programme concerné peut être appelé en langage machine.
Catégorie: instruction
Exemple:

```
10 DEF USR0=1000
20 X=USR0(9*2)
30 END
```

**DELETE[<numéro de ligne>][-<numéro de ligne>]**
Efface toutes les lignes spécifiées. Après l'exécution de cette commande l'ordinateur retourne au mode BASIC.
Catégorie: commande
Exemple: `DELETE 10`

**DIM<nom de tableau>(<indice maximum>[,<indice maximum>][,<indice maximum>])[,<nom de tableau>...]**
Crée de l'espace mémoire pour les tableaux spécifiés et initialise les éléments de tableaux. Quand on fait référence à un tableau qui n'a pas été créé au préalable par une instruction DIM, 10 est pris par défaut comme index maximum. L'index le plus bas est toujours 0. ERASE nous permet de redimensionner un tableau.
Catégorie: instruction
Exemple:

```
10 DIM A(20)
20 FOR K=1 TO 20:A(K)=K:NEXT K
30 FOR K=1 TO 20:PRINT A(K):NEXT K
40 END
```

**DRAW<chaîne>**
Au moyen de DRAW on peut tracer toutes sortes de lignes droites avec des codes simples. Les codes constituent toujours une chaîne.
Exemple:

```
10 SCREEN 2
20 DRAW "D50R100"
30 GOTO 30
```

50 points vers le bas
100 points vers la droite

Donne deux lignes droites qui se joignent. La première ligne fait 50 points vers le bas ('down 50' ou en abréviation D50), et la deuxième 100 points vers la droite (R100).

Pour chaque section de ligne, on peut également indiquer une couleur au moyen de la lettre C et d'un code couleur, par exemple

```
DRAW "C8D50C10R100"
```

La table suivante donne un aperçu des possibilités:

| *Code* | *Signification* |
|---|---|
| **S** | Indique l'échelle. S est suivi d'un nombre. L'échelle correspond alors au nombre /4. |

**A** Après A vient 0, 1, 2 ou 3. Ainsi le système de coordonnées tourne par angles de 90 degrés. L'ordinateur prend A0 par défaut.

**C** Indique la couleur. Voir l'exemple ci-dessus.

**M** Pour dessiner une ligne inclinée. Après M, viennent deux nombres, séparés par une virgule, par exemple

M30,50

Dans cet exemple, une ligne inclinée sera tracée à partir de la dernière position jusqu'au point de coordonnées relatives x = 30 et y = 50. Les valeurs x et y peuvent donc être négatives.

**U** U est l'abréviation de 'up', c'est-à-dire 'en haut'. Par exemple:

U30

veut dire qu'à partir de la dernière position, une ligne doit être tracée vers le haut, sur 30 points d'image.

**D** D est l'abréviation de 'down'. Pour tracer une ligne vers le bas.

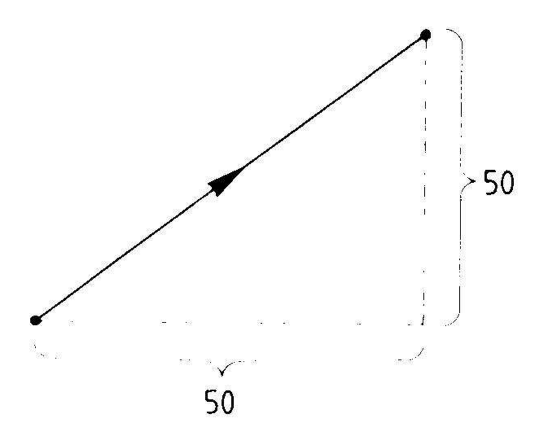
**R** R est l'abréviation de 'right'. Pour tracer une ligne vers la droite.

**L** L est l'abréviation de 'left'. Pour tracer une ligne vers la gauche.

**E** Pour tracer une ligne avec un angle à 45 degrés en haut vers la droite. Exemple:

E50

donne comme résultat:




**F** Pour tracer une ligne avec un angle à 45 degrés en bas vers la droite (voir aussi E).

**G** Pour tracer une ligne avec un angle à 45 degrés en haut vers la gauche (voir aussi E).

**H** Pour tracer une ligne avec un angle à 45 degrés en bas vers la gauche (voir aussi E).

Remarquez que les lignes sont toujours tracées à partir de la dernière position atteinte. Une position de départ fixe peut être obtenue avec BMx,y, (x et y étant les coordonnées de départ).
Exemple:

```
DRAW "BM50,50D30"
```

donne à partir du point (50,50) une ligne sur 30 points vers le bas de l'écran.
On peut également maintenir le dernier point atteint comme point de départ pour une nouvelle ligne, en mettant un N devant le code. On peut également indiquer la chaîne au moyen d'une expression chaîne.
Exemple:

```
....
50 B$="BM50,50D30"
60 DRAW B$
```

Une partie d'une chaîne dans une instruction DRAW peut également être représentée au moyen d'une variable. Dans la partie de l'instruction DRAW, nous mettons la variable X.
Exemple:

```
....
50 B$="BM50,50D30"
60 DRAW "R100XB$"
```

Enfin, on peut indiquer, au moyen d'une variable les nombres figurant dans l'instruction DRAW. La variable se place alors entre les signes = et ;.
Exemple:

```
...
50 K=30
60 DRAW "B10,10R K:"
```

Catégorie: instruction
Exemple:

```
10 SCREEN 2
20 DRAW "BM80,80H20"
30 GOTO 30
```

**DSKF (<numéro de l'unité de disque>)**
Cette fonction indique la mémoire disponible qui reste sur un disque. Les unités sont numérotées comme suit:

| | | | |
|---|---|---|---|
| 0 = unité par défaut | 2 unité B | 4 unité D | 6 unité F |
| 1 = unité A | 3 unité C | 5 unité E | |

Catégorie: fonction
Exemple:

```
10 PRINT DSKF(1)
20 END
```

**DSKI$ (<numéro de l'unité de disque>,<numéro du secteur>)**
Après appel, cette fonction charge le secteur indiqué dans la zone mémoire indiquée par les emplacements &HF351 et &HF352.
Les chiffres 0 à 6 indiquent les unités de disque A à F.
Catégorie: fonction
Exemple:

```
10 SCREEN 0:WIDTH 38:COLOR 15,4,4:DEFINT I
20 PRINT "COPIE SECTEUR PAR SECTEUR"
30 PRINT:PRINT
40 PRINT "introduisez la disquette source dans"
```

```
 50 PRINT "l'unité A:"
 60 PRINT "introduisez la disquette vide dans"
 70 PRINT "l'unite B:"
 80 PRINT "appuyez sur une touche"
 90 A$-INPUT$(1)
100 PRINT:PRINT
110 PRINT "TRACK:"
120 PRINT
130 PRINT "SECTEUR:"
140 FOR I=1 TO 719
150 A$-DSKI$(1,I)
160 LOCATE 8,10:PRINT INT(I/9)
170 LOCATE 8,12:PRINT IMOD9
180 DSKO$ 2,I
190 NEXT I
200 END
```

**DSKO$<numéro d'unité>,<numéro du secteur logique>**
Pour écrire le contenu de la mémoire (indiqué au moyen du contenu des adresses mémoire &HF351 et &HF352) vers le secteur spécifié.
Les chiffres 0 à 6 indiquent les unités de disque A à F.
Catégorie: instruction
Exemple: voir DSKI$

**END**
Termine un programme. Cette instruction peut se trouver n'importe où dans le programme. Un END à la fin d'un programme est optionnel, c'est-à-dire qu'il n'est pas obligatoire. Tous les fichiers encore ouverts seront fermés.
Catégorie: instruction
Exemple:
```
10 INPUT A
20 IF A <5 THEN END
30 END
```

**EOF (<numéro de fichier>)**
Détermine si la fin de fichier (file) a été atteinte.
Catégorie: fonction
Exemple: `IF EOF(2) THEN CLOSE #2`

**ERASE<nom de tableau>[,<nom de tableau>...]**
ERASE permet d'effacer un tableau pour libérer de l'espace de mémoire (voir aussi DIM).
Catégorie: instruction
Exemple:
```
 10 DIM K(100), X$(50)
  ...
500 ERASE K,X$
```

### ERR et ERL

Quand une erreur se produit dans un programme, ERR donne le numéro de l'erreur qui s'est produite (voir l' Annexe des signalisations d'erreur) et ERL donne le numéro de la ligne dans laquelle l'erreur s'est produite.

ERR et ERL peuvent être utilisées dans des programmes de traitement d'erreur écrits par l'utilisateur (voir aussi ON ERROR GOTO), le plus souvent dans des instructions IF...THEN.

Exemple: IF ERR=11 AND ERL=160 THEN...etc.

Catégorie: fonction

Exemple:

```
10 ON ERROR GOTO 50
20 A=25:PRINT A
30 B=A/0
40 END
50 PRINT ERL
60 RESUME NEXT
```

### ERROR<numéro d'erreur>

Affiche sur l'écran le message d'erreur correspondant au <numéro d'erreur> spécifié. Le <numeŕo d'erreur> doit être supérieur à zéro et inférieur à 255. Exemple: introduisez ERROR 11. Sur l'écran s'affiche 'Division by zero'.

BASIC n'utilise pas tous les messages d'erreur compris entre 0 et 255. Vous pouvez utiliser les messages libres pour générer vos propres messages d'erreur.

Catégorie: fonction

Exemple:

```
 10 ON ERROR GOTO 400
 20 INPUT "X:-",A
 30 IF A>100 THEN ERROR 210
    ...
    ...
400 IF ERROR=210 THEN PRINT "100 AU MAXIMUM!"
410 RESUME 20
```

### EXP(<X>)

Calcule l'exponentielle de <X>.

Catégorie: fonction

Exemple: `PRINT EXP(1)`

### FIELD [#]<numéro de fichier>,<largeur de champs> AS <variable chaîne>

Permet l'accès à la mémoire tampon d'un fichier à accès aléatoire. Dans un programme BASIC on peut lire ou écrire dans la mémoire tampon en utilisant les variables de chaîne mentionnées dans la partie AS.

Catégorie: instruction

Exemple: `FIELD#1,20 AS A$, 10 AS B$`

Ainsi les 20 premières positions du tampon du fichier 1 sont attribuées à A$, et les 10 positions suivantes à B$. FIELD ne peut ni lire ni ecrire sur le disque. La somme des largeurs de champs dans une commande FIELD ne peut pas excéder la longueur d'enregistrement du fichier (spécifiée dans la commande OPEN).

**FILES["<nom de fichier>"]**
Donne une liste des fichiers se trouvant sur le disque.
Catégorie: commande
Exemple: `FILES`

**FIX(<X>)**
Donne la partie entière de <X>.
Catégorie: fonction
Exemple: `PRINT FIX(1.7)`

**FOR...NEXT**
La syntaxe complète est:

FOR<variable>=<n>TO<m>[STEP<k>]
instruction 1
instruction 2
...
...
NEXT[<variable>][,<variable>...]

n, m et k doivent être des expressions numériques.
Toutes les instructions entre FOR et NEXT sont exécutées en cascade jusqu'à ce que le compteur (<variable>) dépasse la valeur de m. La valeur initiale du compteur est n et le compteur est augmenté de k chaque fois que toutes les instructions ont été exécutées. Si nous ne spécifions pas de k, le compteur est augmenté chaque fois de 1.
Les instructions FOR...NEXT peuvent comprendre à leur tour d'autres instructions FOR...NEXT.
Catégorie: instruction
Exemple: `10 FOR K=1 TO 10:PRINT K:NEXT K`

**FRE(0) OU FRE(<" ">)**
Si l'argument est un 0, FRE donne la place libre en mémoire. Si l'argument est une chaîne, FRE donne le nombre d'octets libres en mémoire chaîne de caractères. Ce dernier espace peut être adapté avec CLEAR.
Catégorie: fonction
Exemple: `PRINT FRE(0)`

**GET [#]<numéro de fichier>[,<numéro d'enregistrement>]**
Permet de lire un enregistrement sur disque d'un fichier à accès aléatoire dans la mémoire tampon de ce fichier. Le contenu de l'enregistrement peut alors être utilisé dans un programme quand la commande FIELD a été donnée pour le fichier, en utilisant les variables chaîne spécifiées dans cette commande FIELD.
Chaque enregistrement dans un fichier à accès aléatoire a un numéro. Le No d'enregistrement indique l'enregistrement à lire. Si le Numéro d'enregistrement n'est pas precisé GET lit l'enregistrement qui suit l'enregistrement qu'il avait lu ou écrit.
Le numéro d'enregistrement du dernier enregistrement introduit peut être trouvé au moyen de la fonction LOC; voir ci-dessous cette fonction.
Catégorie: instruction
Exemple:
```
10 OPEN "VBD.DAT"AS#1
20 FIELD #1,2 AS A$,10 AS B$
```

```
30 FOR K%=1 TO 10
40 GET #1,K
50 PRINT CVI(A$);B$
60 NEXT
70 CLOSE #1
80 END
```

**GET DATE <X$>[,A]**

Cette instruction permet d'appeler la date gérée par le circuit d'horloge.

Le format dépend de la version et peut se présenter comme suit : MM/JJ/AA ou JJ/MM/AA ou AA/MM/JJ. (M = mois, J = jour, A = année). Lorsque A est introduit, la date d'alarme est renvoyée (l'ordinateur cherche JJ).

Catégorie: instruction

Exemple: `GET DATE PR$:PRINT PR$`

**GET TIME<X$>[,A]**

Cette instruction permet d'appeler l'heure gérée par le circuit d'horloge. Le format est le suivant : HH:MM:SS (H = heure, M = minute, S = secondes). Lorsque A est introduit, l'heure d'alarme est renvoyée.

Catégorie: instruction

Exemple: `GET TIME PR$:PRINT PR$`

**GOSUB<numéro de ligne>**
**RETURN <numéro de ligne>**

Cette instruction permet le saut à un sous-programme. <numéro de ligne> est le numéro de la première ligne du sous-programme. Le sous-programme doit contenir un RETURN pour que l'ordinateur retourne à l'instruction suivant l'instruction GOSUB. Les sous-programmes peuvent renvoyer à leur tour à d'autres sous-programmes.

Catégorie: instruction

Exemple:
```
10 GOSUB 40
20 GOSUB 40
30 END
40 PRINT "SUBROUTINE"
50 RETURN
```

**GOTO <numéro de ligne>**

L'exécution du programme est poursuivie au <numéro de ligne> spécifié.

Catégorie: instruction

Exemple:
```
10 GOTO 30
20 PRINT "A"
30 PRINT "13"
40 END
```

**HEX$(<X>)**

Donne une chaîne qui représente la valeur hexadécimale de <X>. <X> doit être un nombre entier et doit se situer entre −32768 et 65535.

Catégorie: fonction

Exemple: `PRINT HEX$(63)`

**IF...THEN**

La syntaxe complète est:

IF<expression booléenne><opérateur booléen><expression booléenne>...[THEN <numéro de ligne ou instruction(s)>
[ELSE<numéro de ligne ou instruction(s)>]

NOTE: Par 'expression booléenne' on entend 'condition logique', qui peut être vraie ou fausse. Dans les <expressions booléennes> peuvent figurer comme signe relationnel:

| | | | |
|---|---|---|---|
| < | :inférieur à | <= | :inférieur ou égal à |
| > | :supérieur à | >= | :supérieur ou égal à |
| = | :égal à | <> | :différent de |

Si l'on met plusieurs instructions après THEN ou ELSE, celles-ci doivent être séparées par deux points.
L'expression totale ne doit pas dépasser une ligne informatique.
Comme opérateur booléen, on peut utiliser OR, AND, NOT, XOR, EQV ou IMP (voir Annexe F).
Quand l'expression booléenne, entre IF et THEN, est vraie, les instructions suivant THEN sont exécutées. Si THEN n'est pas suivie d'une instruction mais d'un numéro de ligne, l'ordinateur continue à la ligne indiquée.
Si l'expression booléenne entre IF et THEN est fausse, les instructions suivant ELSE sont exécutées, ou un saut à la ligne indiquée est effectué. S'il n'y a pas ELSE l'ordinateur continue à la ligne qui suit immédiatement l'instruction IF...THEN.
Les instructions suivant THEN ou ELSE peuvent de nouveau comprendre une instruction IF...THEN, pourvu que l'ensemble soit sur une seule ligne informatique. Donc l'instruction:

```
IF X>Y THEN PRINT "SUPERIEUR" ELSE IF X<Y THEN PRINT "INFERIEUR" ELSE PRINT
"EGAL"
```

est admise. Chaque ELSE, dans une instruction semblable, est lié au IF le plus proche pour lequel un ELSE correspondant n'a pas encore été trouvé.
Catégorie: instruction
Exemple:

```
10 INPUT A
20 IF A<3 THEN PRINT "A<3"
30 END
```

**INKEY$**

Cette fonction est utilisée sous la forme <expression chaîne>=INKEY$. La fonction vérifie si une touche est enfoncée au clavier. Si oui, le caractère concerné est attribué à <expression chaîne>. Sinon, <expression chaîne> comprend une chaîne vide (dite chaîne zéro).
Catégorie: fonction
Exemple:

```
10 PRINT "TAPEZ UN H: ";
20 A$-INKEY$
30 IF A$<>"H" THEN 20
40 PRINT A$
50 END
```

**INP (<X>)**
Lit l'octet contenu dans le port <X>.
Catégorie: fonction
Exemple:
```
10 A=INP(&HAB)
20 A$="00000000"+BIN$(A):PRINT RIGHT$(A$,8)
```

**INPUT["<texte>";]<variable 1>[,<variable 2>...]**
Permet à l'utilisateur d'entrer des données au clavier. Sur l'écran un point d'interrogation (?) s'affiche toujours.
Quand un <texte> est entré, celui-ci apparaît devant le point d'interrogation sur l'écran.
Les valeurs entrées sont attribuées aux <variables>. Il faut introduire autant de valeurs, séparées par une virgule, qu'il y a de variables dans l'instruction INPUT. Les expressions chaîne sont aussi admises, mais il ne doit pas y avoir de virgule dans la chaîne introduite. S'il y a une erreur dans les valeurs le signal d'erreur? Redo s'affiche et l'instruction INPUT est exécutée de nouveau.
Catégorie: instruction
Exemple:
```
10 INPUT A:PRINT A:END
```

**INPUT#<numéro de fichier>,<variable1>[,<variable 2>...]**
Cette instruction est comparable à INPUT, mais ici il s'agit des valeurs d'un fichier. <numéro de fichier> est le numéro qui est attribué au fichier par OPEN. Les variables du fichier doivent être dans le même ordre que les variables de l'instruction INPUT. Lors de la création d'une chaîne, la fin de la chaîne est déterminée par une virgule, par RETURN ou par Line Feed. Cependant, si le premier caractère de la chaîne est un guillemet ("), un deuxième guillemet est considéré comme la fin de la chaîne.
Catégorie: instruction
Exemple:
```
10 OPEN "CAS:DATA" FOR INPUT AS#1
20 IF EOF(1) THEN 40
30 INPUT#1,X$:PRINT X$:GOTO 20
40 CLOSE#1:END
```

**INPUT$(<X>)**
**INPUT$(<X>,[#]<Y>)**
Cette fonction est utilisée dans la forme <expression chaîne>=INPUT$(<X>). Elle introduit une chaîne de <X> caractères à partir du clavier. Au lieu du clavier, un fichier (Y) peut être indiqué. Tous les caractères sont acceptés, à l'exception de CTRL/C.
Catégorie: fonction
Exemple:
```
10 A$=INPUT$(4):PRINT A$:END
```

**INSTR([I,]<X$>,<Y$>)**
Vérifie si <Y$> est présent dans <X$> et donne alors la position dans <X$>. Si <Y$> ne figure pas dans <X$>, INSTR donne la valeur 0. INSTR cherche à partir du début de <X$>, à moins qu'une autre position de départ soit spécifiée au moyen de [I]. Si <Y$> est une chaîne vide, INSTR donne la valeur 1.
Catégorie: fonction
Exemple:
```
10 A$="ABCDEFG":PRINT INSTR(A$,"BCD"):END
```

**INT(<X>)**
Cette fonction donne la partie entière de <X>.
Catégorie: fonction
Exemple:
```
PRINT INT(3.7)
```

### *INTERVAL ON*
### INTERVAL OFF
### INTERVAL STOP

Permet d'indiquer si une interruption, qui est indiquée par l'horloge incorporée, est validée, annulée ou désactivée. L'interruption doit être indiquée par l'instruction ON INTERVAL GOSUB.
Après l'instruction INTERVAL ON, l'ordinateur saute au sous-programme indiqué dans l'instruction ON INTERVAL GOSUB.
Catégorie: instruction
Exemple:
```
10 ON INTERVAL=300 GOSUB 40
20 INTERVAL ON
30 GOTO 30
40 K=K+6:PRINT K;"SEC"
50 RETURN
```

### KEY <n>,"<chaîne>"

Attribue une <chaîne> à une touche de fonction. <n> est le numéro de la touche de fonction (1 à 10). La <chaîne> doit être entre guillemets (") et peut se composer de 15 caractères au plus. Nous pouvons inclure une commande, par exemple RETURN, dans la <chaîne> en ajoutant +CHR$(13).
(13 code ASCII de RETURN).
Catégorie: instruction
Exemple:
```
KEY 1,"RUN"+CHR$(13)
```
Quand on appuie sur la touche de fonction 1, le programme est immédiatement exécuté, parce que la commande RUN finit par un RETURN.

### KEY LIST

Affiche la programmation des 10 touches de fonction.
Catégorie: instruction
Exemple:
```
10 KEY LIST:END
```

### KEY ON
### KEY OFF

Déterminent si la signification des touches de fonction est affichée ou non sur l'écran.
Catégorie: instruction
Exemple:
```
KEY OFF
```

### KEY(<n>)ON
### KEY(<n>)OFF
### KEY(<n>)STOP

Permet d'activer ou non une touche de fonction. La fonction est activée par ON et annulée par OFF.
Avec STOP, l'ordinateur ne saute pas immédiatement au sous-programme indiqué par ON KEY GOSUB, il ne le fera que quand KEY (n) ON sera donnée.
Catégorie: instruction
Exemple:
```
10 KEY(3) ON
20 ON KEY GOSUB ,, 50
30 GOTO 10
40 END
50 PRINT "KEY3":KEY(3)OFF:RETURN
```

**KILL"<nom de fichier>"**
Pour effacer un fichier se trouvant sur disque.
Catégorie: instruction
Exemple: `KILL "A:X1.DAT"`

**LEFT$(<X$>,<I>)**
Donne une chaîne consistant en les <I> premiers caractères de <X$>. <I> doit se situer entre 0 et 255.
Catégorie: fonction
Exemple: `PRINT LEFT$("MSX",2)`

**LEN(<X$>)**
Donne le nombre de caractères de <X$>. Les espaces sont inclus dans le nombre.
Catégorie: fonction
Exemple: `PRINT LEN("MSX")`

**[LET]<variable>=<valeur>**
Cette instruction attribue une valeur à une variable; le terme LET peut être éventuellement supprimé.
Catégorie: instruction
Exemple:

```
10 LET A=14:PRINT A
20 LET A=A+3:PRINT A:END
```

**LFILES["<nom de fichier>"]**
Donne sur l'imprimante, la liste des fichiers se trouvant sur le disque.
Catégorie: commande
Exemple: `LFILES`

**LINE[[STEP](<X1>,<Y1>)]–[STEP](<X2>,<Y2>)[,<Z>][,B[F[,<OP>]]]**
Trace une ligne entre (X1,Y1) et (X2,Y2). STEP permet d'indiquer que le système de coordonnées est déplacé vers le dernier point indiqué. Z indique la couleur du tracé. Si on indique B, un rectangle de la diagonale correspondante est dessiné. Avec BF, le rectangle est colorié.
<OP> nous permet d'indiquer éventuellement une opération logique (per exemple OR et AND), pour déterminer la couleur. L'effet de cette opération logique est expliqué de manière détaillée dans PSET. Les caractéristiques de chaque mode écran sour détaillées à l'instruction SCREEN
Catégorie: instruction
Exemple:

```
10 SCREEN 5
20 LINE (0,0)-(511,211)
30 GOTO 30
```

**LINE INPUT ["<texte>";]<variable de chaîne>**
Introduit une chaîne qui est tapée sur le clavier. La chaîne peut consister en toutes sortes de caractères et peut avoir n'importe quelle longueur jusqu'à 255 caractères au maximum.
Catégorie: instruction
Exemple: `10 LINE INPUT A$:PRINT A$`

**LINE INPUT#<numéro de fichier>,<variable de chaîne>**
Cette instruction est comparable à LINE INPUT, mais introduit la chaîne à partir d'un fichier. <numéro de fichier> est le numéro qui a été attribué au fichier concerné par OPEN. Une chaîne est entrée dans son ensemble jusqu'à RETURN.
Catégorie: instruction
Exemple:
```
10 OPEN "CAS:DAT" FOR INPUT AS#1
20 IF EOF(1) THEN 50
30 LINE INPUT#1,A$:PRINT A$
40 GOTO 20
50 CLOSE#1:END
```

**LIST [<numéro de ligne>[-[<numéro de ligne>]]]**
Cette commande affiche à l'écran les lignes indiquées. Si l'on n'indique pas de numéros de ligne, le programme entier est affiché.
Catégorie: commande
Exemple: `LIST`

**LLIST[<numéro de ligne>[-[<numéro de ligne>]]]**
Identique à LIST, mais sur l'imprimante.
Catégorie: commande
Exemple: `LLIST`

**LOAD"<dev>:[<nom du programme>]"[,R]**
Pour charger un programme dans la mémoire de l'ordinateur.
Si on utilise le magnétophone à cassettes, le programma doit être sauvegarde sur la cassette au moyen de la commande SAVE. Si vous ajoutez ",R" à la commande, le programme démarrera automatiquement après son chargement.
Catégorie: commande
Exemple: `LOAD"CAS:DEMO"`

**LOC(<numéro de fichier>)**
Cette fonction donne un résultat qui dépend du mode dans lequel le fichier a été ouvert. Si c'est un fichier à accès aléatoire la fonction fournit le numéro du dernier enregistrement lu ou écrit: voir GET. Avec d'autres fichiers la fonction fournit le nombre d'octets traités (lus ou écrits).
Catégorie: fonction
Exemple:
```
10 OPEN "CAS:DAT" FOR OUTPUT AS#1
20 INPUT A$:PRINT#1, A$:PRINT LOC(1)
30 IF A$<>"END" GOTO 20
40 CLOSE#1:END
```

**LOCATE[<X>][,[<Y>][,[<curseur marche/arrêt>]]]**
Déplace le curseur vers la position indiquée. Nous pouvons visualiser ou non le curseur en tapant un 1 ou un 0. X peut varier entre 0 et 79 suivant la valeur fixée par l'instruction WIDTH. Y de 0 à 23.
Catégorie: instruction
Exemple:
```
10 SCREEN 0:CLS:LOCATE 10,10:PRINT "*"
```

**LOF (<numéro de fichier>)**
Cette fonction spécifie la longueur (en octets) du fichier indiqué.
Le fichier indiqué doit avoir été ouvert au préalable.
Catégorie: fonction
Exemple:
```
10 OPEN "A:DAT" FOR INPUT AS#1
20 PRINT LOF(1):CLOSE #1:END
```

**LOG(<X>)**
Donne le logarithme de <X>. <X> doit être supérieur à 0.
Catégorie: fonction
Exemple:
```
PRINT LOG(1 5)
```

**LPOS(<X>)**
X peut être un nombre quelconque. LPOS donne la position de la tête d'impression de l'imprimante.
Catégorie: fonction
Exemple:
```
10 LPRINT:PRINT LPOS (0)
20 LPRINT "MSX";:PRINT LPOS (0)
```

**LPRINT[[USING<format d'impression>;]<expression>]**
Comme PRINT ou PRINT USING, mais avec l'imprimante.
Catégorie: fonction
Exemple:
```
LPRINT "MSX"
```

**LSET<variable chaîne>=<expression chaîne>**
**RSET<variable chaîne>=<expression chaîne>**
Ces commandes permettent d'écrire des données dans le tampon d'un fichier à accès aléatoire. LSET garantit que la chaîne est complétée à droite par des espaces, de sorte que la chaîne est mise à gauche dans le champ. Avec RSET, la chaîne sera justifiée entièrement vers la droite.
Catégorie: fonction
Exemple:
```
 10 MAXFILES=1
 20 OPEN "A:TEST" AS #1
 30 FIELD #1,2 AS N1$, 4 AS N2$, 8 AS N3$, 20 AS N4$
 40 INPUT "A%":A%
 50 INPUT "B!" ;B!
 60 INPUT "C#":C#
 70 INPUT "D$":D$
 80 RSET N1$=MKI$(A%):RSET N2$=MKS$ (B!):RSET N3$=MKD$(C#):
    LSET N4$=D$
 90 PUT #1,1
100 A%=0:B!=0:C#=0:D$=""
110 PRINT A%;B!;C#;D$
120 GET #1,1
130 A%=CVI(N1$):B!=CVS (N2$):C#=CVD(N3$):D$=N4$
140 PRINT A%;B!;C#;D$
150 CLOSE #1
160 END
```

**MAXFILES =<nombre>**
Donne le nombre maximum de fichiers qui peuvent être ouverts simultanément. <nombre> doit être compris entre 1 et 15. Sur une disquette seuls 6 fichiers peuvent être utilisés à la fois.
Catégorie: instruction
Exemple:
```
10 MAXFILES=2
20 OPEN "CAS:DEMO" FOR INPUT AS#1
30 OPEN "LPT:" FOR OUTPUT AS#2
40 INPUT#1,A$
50 PRINT#2,A$
60 CLOSE
```

**MERGE"[<dev>]:[<nom de fichier>]"**
Ajoute un programme stocké dans une mémoire externe au programme présent en mémoire. Le programme doit être enregistré dans le format ASCII.
Les lignes de programme sont placées dans l'ordre ascendant des numéros de ligne. Quand deux lignes ont le même numéro de ligne, la ligne en mémoire est remplacée par la ligne du programme chargé. Le nouveau programme créé reste en mémoire.
Catégorie: commande
Exemple: `MERGE "CAS:PROG1"`

**MID$(<X$>,<I>[<,J>])**
Donne une chaîne qui est une partie de <X$>. La chaîne commence à la position <I> et a une longueur de <J> caractères. Si <J> n'est pas spécifiée, tous les caractères à partir de <I> sont pris.
Catégorie: fonction
Exemple: `PRINT MID$("BASIC",2,3)`

**MID$(<chaîne 1>,<premier caractère>[,<nombre de caractères>])=<chaîne 2>**
Au moyen de cette commande, on peut modifier la <chaîne 1>. A cette fin, nous indiquons à partir de quel caractère de la chaîne 1, on insère la chaine 2.
Catégorie: instruction
Exemple:
```
10 A$= "PHIPLIE"
20 MID$(A$,4,4)="LIPE"
```
La chaîne "PHILIPE" sera attribuée à A$.

**MKI$ (<expression à nombre entier>)**
**MKS$ (<expression réelle>)**
**MKD$ (<expression réelle en double précision>)**
Ces fonctions fournissent une chaîne contenant la version codée du nombre indiqué. La chaîne peut être codée à l'aide des fonctions CVI, CVS et CVD inversées; voir sous ces fonctions.
Les chaînes qu'on obtient en appliquant ces fonctions, peuvent être écrites dans un enregistrement d'un fichier à accès aléatoire.
Catégorie: fonction
Exemple:
```
10 RSET D$:MKD$(WD#)
20 RSET S$:MKS$(WS!)
30 RSET I$:MKI$(WI%)
```

**MOTOR ON**
**MOTOR OFF**
Permet la télécommande d'un magnétophone à cassettes. MOTOR ON met l'appareil en service et MOTOR OFF l'arrête. Les termes ON et OFF peuvent éventuellement être omis. On passe alors d'un état à l'autre.
Catégorie: instruction
Exemple:

```
10 MOTOR ON
20 CLOAD "DEMO"
```

**NAME "[<dev:>]<ancien nom>" AS "<nouveau nom>"**
Pour changer le nom d'un fichier sur une disquette.
Catégorie: instruction
Exemple: `NAME "VBD" AS "EXP1"`

**NEW**
A pour résultat d'effacer le programme présent en mémoire.
Toutes les variables sont effacées et tous les fichiers ouverts sont fermés.
Catégorie: commande
Exemple: `NEW`

**OCT$<X>**
Donne une chaîne qui présente la valeur octale de <X>. <X> est arrondi d'abord à un nombre entier.
Catégorie: fonction
Exemple: `PRINT OCT$(636)`

**ON ERROR GOTO <numéro de ligne>**
Ainsi, une erreur qui s'est produite dans le programme, ne sera pas signalée sur l'écran, comme d'habitude, mais le programme sera poursuivi au <numéro de ligne> indiqué. Ce-ci offre la possibilité d'écrire ses propres programmes de traitement d'erreur. <numéro de ligne> est la première ligne du programme de traitement d'erreur. Dans le programme, on peut utiliser les variables ERR et ERL.
Avec RESUME, on retourne du programme de traitement d'erreur vers le programme principal. L'instruction peut être supprimée par ON ERROR GOTO 0. Il est recommandé d'exécuter, dans un programme de traitement d'erreur, l'instruction ON ERROR GOTO 0 pour le traitement d'erreurs imprévues.
Catégorie: instruction
Exemple:

```
10 ON ERROR GOTO 50
20 INPUT "TEXTE";A$
30 IF LEN(A$)>5 THEN ERROR 250
40 END
50 IF ERR=250 THEN PRINT "TEXTE TROP LONG":RESUME 20
60 ON ERROR GOTO 0
70 END
```

**ON <expression nombre entier> GOTO <numéro de ligne>[,<numéro de ligne>...]**
Permet de continuer l'exécution du programme à un numéro de ligne qui est déterminé par la valeur de <l'expression nombre entier>. Si <l'expression nombre entier> fournit par exemple la valeur 3, le programme sera poursuivi au troisième numéro de ligne indi-qué après GOTO.

La valeur de <l'expression nombre entier> ne peut pas être négative, ou supérieure à 255. Remarque: si cette valeur est 0 ou supérieure au nombre de numéros de ligne indiqué, l'ordinateur poursuit le programme par la première instruction exécutable qui suit.
Catégorie: instruction
Exemple:
```
10 INPUT N
20 ON N GOTO 30,40
30 PRINT "1":GOTO 50
40 PRINT "2":GOTO 50
50 END
```

**ON <expression nombre entier> GOSUB <numéro de ligne>[,<numéro de ligne>...]**
Fonctionne de la même façon que l'instruction ON...GOTO, mais les numéros de ligne indiqués doivent être la première ligne d'un sous-programme.
Catégorie: instruction
Exemple:
```
10 INPUT N
20 ON N GOSUB 40,50
30 GOTO 60
40 PRINT "1":RETURN
50 PRINT "2":RETURN
60 END
```

**ON INTERVAL=<temps> GOSUB <numéro de ligne>**
Indique qu'après un temps d'attente indiqué par INTERVAL, le programme doit sauter au sous-programme indiqué.
Catégorie: instruction
Exemple: voir INTERVAL ON/OFF/STOP

**ON KEY GOSUB <numéro de ligne>[,<numéro de ligne>...]**
Indique quel sous-programme doit être exécuté, quand une touche de fonction est enfoncée. Chaque numéro de ligne correspond dans l'ordre aux touches 1 à 10.
Catégorie: instruction
Exemple: voir KEY (<n>) ON/OFF/STOP

**ON SPRITE GOSUB <numéro de ligne>**
Indique vers quel sous-programme le saut doit être fait quand deux sprites se recontrent.
Catégorie: instruction
Exemple: voir SPRITE ON/OFF/STOP

**ON STOP GOSUB <numéro de ligne>**
Indique vers quel sous-programme le saut doit être fait quand le programme est interrompu avec CTRL/STOP.
Catégorie: instruction
Exemple: voir STOP ON/OFF/STOP

**ON STRIG GOSUB <numéro de ligne>[,<numéro de ligne>...]**
Indique vers quel sous-programme le saut doit être fait, quand un bouton ou la barre espace est enfoncé. Le bouton ou la barre d'espace concerné doit être activé au moyen de l'instruction STRIG (<n>) ON.

| *n.* | *indique* |
|---|---|
| **0** | barre d'espacement |
| **1** | manette de jeu 1, bouton 1 |
| **2** | manette de jeu 2, bouton 1 |
| **3** | manette de jeu 1, bouton 2 |
| **4** | manette de jeu 2, bouton 2 |

Catégorie: instruction
Exemple: voir STRIG (<X>) ON/OFF/STOP

**OPEN "<dev>:[<nom de fichier>]" FOR <traitement> AS [#]<numéro>**
Pour ouvrir un fichier.

Pour <nom de fichier>, on peut utiliser un nom se composant de 8 caractères au maximum et une extension de trois caractères (à l'exception du magnétophone à cassettes; dans ce cas le nom ne peut compter que 6 lettres).
Quand, pour 'traitement' on prend OUTPUT, on écrit des données dans le périphérique considéré. Avec INPUT, on lit les données depuis ce périphérique.
Si For <traitement> est omis, il s'agit d'un fichier d'accés aléatoire.

Outre OUTPUT et INPUT, on peut donner APPEND. Dans ce cas, il s'agit de l'extension d'un fichier aléatoire. Le tableau suivant donne un aperçu des possibilités.

| *<dev>* | *OUTPUT* | *INPUT* | *APPEND* | *Omission de FOR <traitement>* |
|---|---|---|---|---|
| CRT | * | | | |
| GRP | * | | | |
| LPT | * | | | |
| CAS | * | * | | |
| MEM | * | * | * | |
| A à F | * | * | * | * |
| COM | * | * | | * |

L'expression AS[#]<chiffre> octroye un numéro au fichier, qui est ensuite utilisé dans des instructions comme INPUT#, PRINT# etc. Signalons enfin qu'avec l'instruction OPEN, on suppose que le périphérique <dev> indiqué est réellement raccordé. Dans le cas contraire, un message d'erreur s'affiche.
Catégorie: instruction
Exemples: voir CLOSE, GET, INPUT# et LINE INPUT#.

**OUT <numéro de port, expression>**
La donnée indiquée par <expression> est dirigé vers le port indiquée par <numéro de port>.
Ces deux données doivent être comprises entre 0 et 255.
Catégorie: instruction
Exemple: `10 OUT &HAB.INP(&HAB)`

**PAD(<X>)**
Cette fonction importante permet de vérifier quel est le statut de la tablette graphique, crayon lumineux, souris ou 'track ball'.

Le tableau suivant montre quel est l'appareil indiqué.

| *X* | *est utilisé pour* |
|---|---|
| 0 à 3 | tablette graphique raccordée au connecteur de poignée de jeu 1 |
| 4 à 7 | tablette graphique raccordée au connecteur de poignée de jeu 2 |
| 8 à 11 | crayon lumineux |
| 12 à 15 | Souris ou 'track ball' sur le connecteur de poignée de jeu 1 |
| 16 à 19 | Souris ou 'track ball' sur le connecteur de poignée de jeu 2 |

Pour la tablette graphique, on utilise le tableau suivant:

| *X* | *signification de la valeur de la fonction PAD(X)* |
|---|---|
| 0 ou 4 | détermine si la tablette est touchée 0 (valeur non disponible) ou −1 (valeur disponible) |
| 1 ou 5 | coordonnée X de l'endroit touché; PAD(0) ou PAD(4) doit être −1. |
| 2 ou 6 | coordonnée Y de l'endroit touché; PAD(0) ou PAD(4) doit être −1. |
| 3 ou 7 | bouton de validation (0 = non enfoncé et −1 = enfoncé) |

Pour le crayon lumineux, on utilise le tableau suivant:

| *X* | *Signification de la valeur de la fonction PAD(X)* |
|---|---|
| 8 | Détermine si le crayon est prêt ou non. 0 (non prêt) ou −1 (prêt) |
| 9 | coordonnée X:PAD(8) doit d'abord être −1. |
| 10 | coordonnée Y:PAD(8) doit d'abord être −1. |
| 11 | détermine si l'interrupteur est enfoncé. (0 = non et −1 = oui) |

Pour la souris et le track ball, on utilise le tableau suivant:

| *X* | *Signification de la fonction* |
|---|---|
| 12 ou 16 | donne toujours −1, mais doit être appelé pour déterminer X ou Y. |
| 13 ou 17 | coordonnée X |
| 14 ou 18 | coordonnée Y |
| 15 ou 19 | aucune signification |

Remarquons que chaque fois qu'une coordonnée X et Y doit être lue, il faut d'abord contrôler un statut spécifique. Ainsi, PAD(1) ne donnera la valeur de la coordonnée X que si PAD(0) à donné la valeur −1. Dès que le statut spécifique est atteint, il faut lire les coordonnées aussi vite que possible. La fonction STRIG nous permet de déterminer le statut de la souris ou du track ball.
Catégorie: fonction
Exemple:
```
10 SCREEN 2
20 AA=0
30 IF PAD(0)=0 THEN 20
40 X=PAD(1):Y=PAD(2)
50 IF AA=0 THEN PSET(X,Y) ELSE LINE -(X,Y)
60 AA=1
70 GOTO 30
```

**PAINT [STEP](<X,Y>)[,[<couleur surface>][,<couleur limite>]]**

Indique qu'une certaine région doit être coloriée. Par STEP, le système de coordonnées est déplacé. Par x et y, nous indiquons un point. La surface dans laquelle ce point se trouve est coloriée. Quand la ligne de limite n'est pas complètement ininterrompue, l'écran entier est colorié.

Dans les modes écran 2 et 4, la <couleur de surface> doit être identique à la <couleur limite>. Dans ce cas, on ne peut donner de valeur pour <couleur limite>. Dans les modes 3, 5, 6, 7 et 8, il peut y avoir une différence dans ces indications de couleur.

Dans les modes écran 2. 3, 4, 5 et 7, les codes couleur doivent être compris entre 0 et 15. Dans le mode écran 6 le code couleur peut varier de 0 à 3 et de 0 à 255 dans le mode 8 (voir COLOR).

Catégorie: instruction

Exemple:
```
10 SCREEN 2:COLOR 15,4,4
20 CIRCLE (180,180),40,8
30 PAINT (180,180),2,8
40 GOTO 40
```

**PDL(<X>)**

Donne la position de la pédale (<X> peut être 1 à 12). La valeur varie de 0 à 255. Pour <X> = 1,3,5,7,9 ou 11 l'ordinateur suppose que le connecteur de manette 1 est utilisé. Si <X> = 2,4,6,8,10 ou 12 le connecteur 2 est utilisé.

Catégorie: fonction

Exemple: `10 PRINT PDL(1):GOTO 10`

**PEEK(<X>)**

Donne le contenu de l'adresse mémoire X. En décimal, X doit se situer entre −32768 et 65536. Si X est négatif, le complément à 2 est utilisé: PEEK (-1) = PEEK (65536-1).

Catégorie: fonction

Exemple: `PRINT PEEK(65535)`

**PLAY [<premier canal>][,[<deuxième canal>][,<troisième canal>]]**

Instruction permettant de produire du son selon certaines indications. PLAY 'CDE', par exemple, donne les notes do ré mi. Devant l'indication d'une note (C,D,E,F,G,A,B et aussi C+ pour C diese, etc.), on peut mettre des lettres et des nombres qui ont une signification spéciale:

*Code* — *signification*

On — Indique l'octave, par exemple PLAY "O5CDE": veut dire: jouez do ré mi de l'octave 5. "n" doit être compris entre 1 et 8. La valeur par défaut est 4.

Ln — Indique la durée d'une note, selon:

| durée | code |
|---|---|
| 4 Mesures | L1 |
| 2 Mesures | L2 |
| 1 Mesures | L4 |
| 1/2 Mesures | L8 |
| 1/4 Mesures | L16 |
| 1/8 Mesures | L32 |
| 1/16 Mesures | L64 |

Par défaut, "n" est égal à 4.

| | |
|---|---|
| Nn | Joue une note comprise entre 1 et 96. |
| A à G | Indique la note. Si nous donnons par exemple A5, cela correspond à la du cinquième octave. |
| Rn | Indique un silence, selon: |

| durée | code |
|---|---|
| 4 Mesures | R1 |
| 2 Mesures | R2 |
| 1 Mesures | R4 |
| 1/2 Mesures | R8 |
| 1/4 Mesures | R16 |
| 1/8 Mesures | R32 |
| 1/16 Mesures | R64 |

| | |
|---|---|
| Vn | Indique le volume: N=15 correspond au 'volume maximum'. Par défaut "n" = 8. |
| • | prolonge la durée d'une note par un facteur 1,5. |
| Sn | Indique la forme (0≤n≤15). Par défaut, "n" est égal à 1. |
| Tn | Indique le tempo. la valeur de n détermine le nombre de mesures par minute. n peut se situer entre 32 et 255. La valeur par défaut est 120. |
| Mn | Définit la période de l'enveloppe. Par défaut, "n" est égal à 255. |

Catégorie: instruction
Exemple: `PLAY "CDECDEEFGC"`

**PLAY(<X>)**
Indique les canaux sonores utilisés: 0 = tous les canaux, 1 = canal 1, 2 = canal 2, 3 = canal 3. Si le canal sonore indiqué est activé, PLAY attribue la valeur -1. Dans le cas contraire, PLAY prend la valeur 0.
Catégorie: fonction
Exemple:
```
10 PRINT PLAY(0)
20 PLAY "CDEFG"
30 PRINT PLAY(0):END
```

**POINT(<X>,<Y>)**
Donne la couleur du point indiqué par les coordonnées X,Y. Le champ de X et Y dépend du mode graphique utilisé (voir SCREEN).
Catégorie: fonction
Exemple: `PRINT POINT(50,50)`

**POKE <adresse, donnée>**
Range la donnée à l'adresse mémoire indiquée. <adresse> peut prendre une valeur allant de -32768 à 65535. <donnée> est compris entre 0 et 255.
Catégorie: instruction
Exemple: `POKE 5000,100`

**POS(0)**
Donne la position du curseur sur la ligne. 0 est un faux argument et n'a pas de signification. La position la plus à gauche est la position 0.
Catégorie: fonction
Exemple: `10 SCREEN 0:LOCATE 10,20:PRINT POS(0):END`

**PRESET [STEP](<x,y>)[,<couleur>[<operation>]]**

Met ou efface un point (x,y) de l'écran graphique. Quand une couleur est indiquée, l'effet de PRESET est identique à celui de PSET. Sans cette indication, un point déja indiqué est effacé. Au moyen de STEP, on peut déplacer le système de coordonnées.
L'instruction <opération> permet de modifier encore la couleur. On trouvera une énumération complète de toutes les possibilités à PSET.
Catégorie: instruction
Exemple:

```
10 SCREEN 2
20 FOR K=1 TO 100:PRESET (K,K),1:NEXT
30 FOR K=1 TO 50:PRESET (K,K):NEXT
40 GOTO 40
```

**PRINT [<expression>[,ou;]][<expression><,ou;>...]]**

Affiche à l'écran la valeur de variables, d'expressions numériques ou de chaînes. Les chaînes doivent être écrites entre guillemets (").
La position est déterminée par le signe de séparation. Si ce signe de séparation est un point virgule (;) ou un espace, les données sont mises directement l'une après l'autre. Si le signe de séparation entre deux données est une virgule (,), la deuxième donnée est mise dans la zone suivante.
BASIC divise une ligne en zones de 14 colonnes. Si l'on termine l'instruction PRINT par une virgule ou par un point virgule, les données dans l'instruction suivante PRINT sont mises sur la même ligne, sinon sur la ligne suivante. On peut également indiquer le terme PRINT par un signe d'interrogation, par exemple: `10 ? "RESULTAT"; A`
Catégorie: instruction
Exemple: `10 A=2:PRINT"MSX-";A`

**PRINT USING "<format>";<expression>[; <expression>...]**

Cette instruction est une extension de l'instruction PRINT, grâce à laquelle on peut indiquer soi-même, au moyen d'un <format>, sous quelle forme on désire afficher les données. Les <expressions> doivent être séparées par un point virgule (;).

Pour afficher des chaînes, on peut choisir l'un des trois <codes de notation> suivants:

**!** — Seul le premier caractère de la chaîne est affiché.

**\n espacements** — Maintenant les premiers 2+n caractères de la chaîne sont affichés.

**&** — Ce signe indique qu'à cette place le contenu de la variable chaîne donnée doit être affiché.

Pour afficher des valeurs numériques, on peut choisir parmi les codes de notation spéciaux suivants:

# — Donne le nombre de positions avant et après le point.
Les nombres trop grands sont arrondis. Les nombres trop petits sont adaptés au moyen de zéros et d'espaces. Si le nombre est trop grand pour le format spécifié, le signe de pourcentage (%) précède le nombre.

Exemples:

```
PRINT USING "##.##";1.2345
1.23
PRINT USING "##.##";99.996
%100.00
```

\+ Un signe plus au départ ou à la fin du <format> garantit qu'un signe plus ou un signe moins sera affiché avant ou après un nombre.

– Un signe moins à la fin du <format> garantit qu'un signe moins sera affiché après les nombres négatifs.

** Un double astérisque placé en tête de format provoque le remplissage par des astérisques des positions inoccupées à gauche du nombre imprimé. En outre, ces 2 astérisques spécifient des positions pour 2 chiffres supplémentaires.

**$$** Un signe dollar en tête de format garantit qu'un signe dollar est affiché devant le nombre. La notation scientifique (voir plus loin) ne peut pas être utilisée en combinaison avec $.

**$ Donne une combinaison des deux effets précédents.

, Une virgule à la gauche du point décimal assure qu'une virgule est affichée toutes les trois positions.

Exemple: `PRINT USING "####,.##";1234 5`

`1,234.50`

Une virgule à la fin du <format> est affichée normalement.

Exemple: `PRINT USING "####,.##";1234 5`

`1,234.50`

^^^^ En mettant quatre flèches à la fin du <format>, le nombre est affiché en notation scientifique.

Examples: `PRINT USING "##.## ^^^^";123.45`

`1.23E+0.2`

**texte** Un texte en fin ou en tête de format, peut être affiché.

Exemple: `PRINT USING "FR ##.## FRANC";12.34`

`FR 12.34 FRANC`

Categorie: instruction
Exemple: voir ci-dessus

**PRINT # et PRINT # USING**

La syntaxe complète est:

```
PRINT #<numéro de fichier>,[USING"<format>";]<expression>[;expression>...]
```

Pour stocker des données dans un fichier. <numéro de fichier> est le numéro qui a été attribué au fichier par OPEN. Pour le <format>, mêmes possibilités que celles décrites pour l'instruction PRINT USING. Les <expressions> sont stockées dans le fichier l'une après l'autre, et, comme signe de séparation, on ne peut utiliser que le point-virgule (;). Les chaînes sont mises l'une après l'autre dans le fichier, et, pour cette raison, elles ne sont plus reconnaissables comme chaînes individuelles. Mettons par exemple que A$=JEAN et B$=PIERRE. L'instruction est alors:

```
PRINT #1,A$;B$
```

JEANPIERRE est enregistrée dans le fichier. Ceci ne peut être récupéré que comme une seule chaîne. Ce problème peut être évité en ajoutant des signes de séparation. Par exemple:

```
PRINT #1,A$;",";B$
```

donne le résultat: JEAN,PIERRE

Une autre possibilité est d'écrire les chaînes dans le fichier avec deux instructions:

```
PRINT #1,A$
PRINT #1,B$
```

En ne finissant pas l'instruction PRINT par un point-virgule, un RETURN est ajouté automatiquement en fin de la chaîne. Le RETURN fonctionne maintenant comme signe de séparation. Quand la chaîne elle-même comprend une virgule, elle est considérée comme deux chaînes. Par exemple, A$=JEAN,PIERRE et l'instruction

```
PRINT #1,A$
```

donne dans le fichier JEAN,PIERRE et, après entrée avec:

```
INPUT #1,A$,B$
```

donnera comme résultat que A$= JEAN et B$= PIERRE. On peut éviter ce problème en mettant la chaîne entre guillemets(") dans le fichier (voir également l'instruction INPUT). L'instruction devient maintenant:

```
PRINT #1,CHR$(34);A$;CHR$(34).
```

34 est le code ASCII des guillemets (").
Catégorie: instruction
Exemple: voir ci-dessus

**PSET [STEP](<x,y>)[,<Z>[,<opération>]]**
Ceci permet de placer un point sur l'écran graphique. Les coordonnées de ce point correspondent à X,Y. Les valeurs que peuvent prendre X et Y dépendent du mode graphique choisi (voir SCREEN). Z indique le code couleur. Dans les modes écran 2, 3, 4, 5 et 7, Z peut varier de 0 à 15; dans le mode écran 6, Z peut varier de 0 à 3 et dans le mode 8, de 0 à 255 (voir COLOR). Dans les modes 5 à 8 on peut aussi faire dépendre la couleur d'une opération donnée.
Le tableau suivant montre comment le code couleur C est déterminé par l'opération logique entre le code couleur donné (Z) et le code couleur (S) du point d'écran indiqué.

| *Opération logique* | *C est déterminé par l'opération logique suivante:* |
|---|---|
| XOR | C=NOT(Z)*S+Z*NOT(S) |
| OR | C=Z+S |
| AND | C=Z*S |

Voir annexe F pour les opérations logiques.
De plus on peut également utiliser les indications TXOR, TOR, TAND qui ont le même effet que les indications correspondantes sans le 'T', à la différence que la couleur transparente n'exerce aucun effet.
Signalons encore que l'effet de XOR, OR et AND se détermine simplement avec une instruction PRINT. Par exemple pour C=1, Z=2 et AND; PRINT 1 AND 2.

Catégorie: instruction
Exemple:

```
10 SCREEN 2
20 FOR K=1 TO 100:PSET(K,K):NEXT
30 GOTO 30
```

**PUT [#]<numéro de fichier>[,<numéro d'enregistrement>]**
Permet d'écrire dans le fichier l'enregistrement se trouvant dans la mémoire tampon du fichier indiqué. Le numéro d'enregistrement indique l'enregistrement à écrire. Si le numéro d'enregistrement n'est pas precisé, PUT ecrit l'enregistrement à la suite du dernier enregistrement écrit ou lu.
Voir aussi FIELD, GET, LSET, RSET et OPEN
Catégorie: instruction
Exemple:

```
 10 OPEN "EXPL.DAT" AS#1
 20 FIELD #1,2 AS A$,10 AS B$
 30 FOR K%=1 TO 10
 40 INPUT N%,S$
 50 LSET A$=MKI$(N%)
 60 RSET B$=S$
 70 PUT #1,K%
 80 NEXT
 90 CLOSE #1
100 END
```

**PUT SPRITE<numéro de plan>[,[STEP] (x,y)][,<couleur>][,<numéro de sprite>]**
met le sprite avec la couleur et le numéro indiqués à la position (x,y). <numéro de plan> est la priorité du sprite. 0 est la priorité la plus haute; 31 est la priorité la plus basse. Si deux sprites se trouvent à la même position sur l'écran, le sprite avec la priorité la plus haute est visible.
Avec STEP, on peut éventuellement déplacer le système de coordonnées. Pour un sprite 8×8, on peut prendre comme <numéro de sprite> un nombre entre 0 et 255. Pour un sprite 16×16 un nombre entre 0 et 63.
L'instruction COLOR SPRITE vous permet de modifier la couleur du sprite. Dans les modes écran 2 et 3, on ne peut reproduire que 4 sprites les uns à côté des autres (sur une ligne). Dans les modes écran 4, 5, 6, 7 et 8, on peut en placer 8 côte à côte.
Catégorie: instruction
Exemple:

```
10 CLS:COLOR,11,11:SCREEN 2
20 A$="":FOR K=1 TO 8:A$=A$+CHR$(16):NEXT
30 SPRITE$(1)=A$:PUT SPRITE 0,(40,40),1,1
40 GOTO 40
```

**READ<variable>[,<variable>...]**
READ est toujours utilisée en combinaison avec DATA et attribue les valeurs spécifiées par DATA aux <variables> indiquées.
Catégorie: instruction
Exemple: voir DATA

**REM<commentaire>**
Permet d'ajouter un commentaire à un programme. BASIC ignore toute informations après REM. REM peut être abrégée par le symbole'.
Catégorie: instructions

Exemple:
```
10 REM BEETHOVEN
20 PLAY "GR8GR8GR8L2D+"
```

**RENUM [<nouveau numéro de ligne>][,[<ancien numéro de ligne>][,[<pas de progression>]]]**

Sert à renuméroter les lignes. <nouveau numéro de ligne> est le nouveau numéro de la première ligne. S'il n'est pas spécifié, l'ordinateur commence par 10.
<ancien numéro de ligne> indique à partir de quelle ligne la renumérotation doit commencer. S'il n'est pas spécifié, l'ordinateur commence par la première ligne.
<pas de progression> est le nombre par lequel le numéro de ligne doit être augmenté. Le pas de progression par défaut est 10.
Catégorie: commande
Exemple:
```
RENUM 50,10,20
```

**RESTORE [<numéro de ligne>]**

Permet de lire dès le début, au moyen de READ, des listes avec des valeurs spécifiées par DATA. Voir aussi DATA. <Numero de ligne> indique à partir de quelle ligne, on peut lire ou relire une ligne concernée par DATA.
Catégorie: instruction
Exemple:
```
10 READ A, B, C:PRINT A, B, C
20 RESTORE
30 READ D,E:PRINT D, E
40 DATA 5, 6, 7
```

**RIGHT$(<X$>,<I>)**

Donne une chaîne qui contient les <I> derniers caractères de <X$>.
Catégorie: fonction
Exemple:
```
PRINT RIGHT$("MSX",2)
```

**RND(<X>)**

Donne un nombre aléatoire compris entre 0 et 1. La série de nombres aléatoires générée dépend de la valeur de X.

| | |
|---|---|
| **<X> = négatif** | l'ordinateur commence au début de la série de nombres aléatoires; |
| **<X> = positif** | donne le nombre aléatoire suivant dans la série |
| **<X> = 0** | donne de nouveau le dernier nombre aléatoire. |
| | Par RND (-TIME) on obtient vraiment d'autres nombres chaque fois (comparez RANDOMIZE à d'autres versions BASIC). |

Catégorie: fonction
Exemple:
```
10 FOR K=1 TO 100:PRINT RND(1):NEXT:END
```

**RSET**

voir LSET

**RUN[<X>]**
**RUN"[<dev>:]<nom du programme>"[,R]**

La commande RUN permet de lancer un programme. RUN X indique qu'un programme présent dans la mémoire doit être exécuté à partir de la ligne X. En donnant <dev>, nous indiquons que le programme se trouve dans un périphérique de stockage. Pour <dev>,

nous pouvons mettre: CAS, MEM, A à F et COM[<n>].
RUN a pour conséquence que tous les fichiers restés ouverts se ferment, sauf si R est donné.
Catégorie: commande
Exemple: `RUN`

**SAVE"[<dev>:]<nom du programme>"[,A]**
Cette commande permet de mémoriser sur un périphérique un programme qui se trouve en mémoire (voir LOAD). Pour <dev>, on peut mettre: CAS, MEM, A à F et COM[<n>].
Le nom du programme est le nom que nous devons utiliser avec LOAD et MERGE pour appeler à nouveau le programme. Nous indiquons par A qu'un fichier doit être mémorisé dans le format ASCII.
Catégorie: commande
Exemple: `SAVE "CAS:DATA"`

**SCREEN[<X>[,<Y>[,<Z>[,<XX>[,<YY>[,<ZZ>]]]]]]**
Cette instruction SCREEN indique le mode d'utilisation de l'écran. A la mise en route de l'ordinateur, les valeurs par défaut sont SCREEN 0,0,1,1,0,0.
Les lettres ont la signification suivante:

| *<X>* | *Mode Texte ou Graphique* |
|---|---|
| 0 | Mode texte 1 |
| | avec WIDTH 40 (40 col. x 24 lignes) |
| | avec WIDTH 80 (80 col. x 24 lignes) |
| 1 | Mode Texte 2 |
| | avec WIDTH 32 (32 col. x 24 lignes) |
| 2 | Mode graphique 1 (256x192 points) |
| 3 | Mode graphique 2 (64x48 points) |
| 4 | Mode graphique 3 (256x192 points) |
| 5 | Mode graphique 4 (256x212 points) |
| 6 | Mode graphique 5 (512x212 points) |
| 7 | Mode graphique 6 (512x212 points) |
| 8 | Mode graphique 7 (256x212 points) |

Le nombre de couleurs différentes sur un écran dépend du mode SCREEN comme suit:

| | | *Nb. de Couleurs/écran* |
|---|---|---|
| Mode texte | 1 | 2 parmi 512 |
| Mode texte | 2 | 2 parmi 512 |
| Mode graphique | 1 | 16 parmi 512 |
| Mode graphique | 2 | 16 parmi 512 |
| Mode graphique | 3 | 16 parmi 512 |
| Mode graphique | 4 | 16 parmi 512 |
| Mode graphique | 5 | 4 parmi 512 |
| Mode graphique | 6 | 16 parmi 512 |
| Mode graphique | 7 | 256 |

Les instructions suivantes peuvent uniquement être utilisées dans un mode graphique: CIRCLE, COLOR SPRITE, COLOR SPRITE$, COPY, DRAW, LINE, PAINT, PSET, PRESET, ON SPRITE GOSUB, SPRITE ON/OFF/STOP, POINT et PUT SPRITE.

<Y> *Format des sprites*
0 8x8, sans agrandissement
1 18x8, avec agrandissement (facteur 2)
2 16x16, sans agrandissement
3 16x16, avec agrandissement (facteur 2)

<Z> *Signal sonore ou non en appuyant sur une touche*
0 Pas d'émission d'un signal sonore
1 Emission d'un signal sonore

<XX> *Vitesse à l'enregistement de la cassette*
1 1200 bauds
2 2400 bauds

<YY> *Type d'imprimante*
0 Imprimante MSX
1 Autre

<ZZ> *Mode display (affichage)*
0 Normal
1 Entrelacé (même page)
2 Normal: succession de pages paires et impaires
3 Entrelacé: succession de pages paires et impaires

Dans les modes affichage 2 et 3, le numéro des pages à afficher doivent être impair. La page paire qui est affichée (en alternance), s'obtient à partir de la page 1 (voir SET PAGE).

Catégorie: instruction
Exemple: voir COLOR SPRITE, LINE, PSET

**SET ADJUST(<X>,<Y>)**
Cette commande permet de recadrer l'image sur l'écran. Ceci peut être important si l'image de votre moniteur ou téléviseur n'est pas bien centrée. X et Y peuvent varier entre -7 et 8. Les valeurs données sont stockées et conservées dans le circuit d'horloge, même lorsqu'on coupe le courant.
Catégorie: instruction
Exemple: `SET ADJUST (3,2)`

**SET BEEP<X>,<Y>**
Cette instruction permet de déterminer la tonalité du signal sonore que l'ordinateur fait entendre dans de nombreuses situations. X peut varier de 1 à 4 et détermine le type de son. Y détermine le volume et peut varier de 1 à 4.
Les valeurs sont conservées dans le circuit d'horloge même lorsque l'ordinateur est débranché.
Catégorie: instruction
Exemple: `SET BEEP 3,2`

**SET DATE <X$>[,A]**
Cette instruction permet de fixer la date. Le format en est le suivant:

MM/JJ/AA ou
JJ/MM/AA ou
AA/MM/JJ

car il dépend de la version MSX qui varie en fonction du pays de commercialisation. Nous pouvons indiquer avec A qu'un signal sonore doit être produit (en combinaison avec SET TIME...,A). Si A est donné, l'ordinateur ne va chercher que le jour.
La chaîne donnée est stockée dans le circuit d'horloge et la date est conservée en permanence même lorsque l'ordinateur est débranché.
Catégorie: instruction
Exemple: `SET DATE "12/02/86"`

**SET PAGE <X>,<Y>**
Pour comprendre cette instruction, nous devons savoir que la mémoire vidéo est divisée en une série de pages. Le nombre de pages dépend des modes graphiques sauf indication contraire, la page 0 est affichée. Avec SET PAGE X,Y, nous indiquons que la page X doit être reproduite et que les instructions avec lesquelles nous construisons une nouvelle image sont stockées dans la page Y. Cette instruction ne s'applique que dans les modes écran 5, 6, 7 et 8. Pour une mémoire vidéo de 128 Ko, on utilise les conventions suivantes:

| *Mode* | *Pages* |
|---|---|
| 5 | 0, 1, 2 et 3 |
| 6 | 0, 1, 2 et 3 |
| 7 | 0 et 1 |
| 8 | 0 et 1 |

Catégorie: instruction
Exemple:
```
10 SCREEN 5:CLS
20 LINE(0,0)-(100,100),1,BF
30 SET PAGE 0,1:LINE(100,100)-(200,200),1,BF
40 SET PAGE 1,0:SET PAGE 0,1:GOTO 40
```

**SET PASSWORD <X$>**
Cette instruction permet d'attribuer un mot de passe à l'ordinateur. Cette donnée est conservée même lorsque l'ordinateur est débranché. Lorsqu'on a donné un 'mot de passe' à l'ordinateur, celui-ci le demande lors de la mise en route. Ceci permet d'éviter que des personnes non autorisées se servent de l'ordinateur. X$ représente le mot de passe; il s'agit d'une chaîne de 255 caractères au maximum.
L'ordinateur ne conserve qu'une seule des trois instructions SET PASSWORD, SET PROMPT et SET TITLE, à savoir celle qui correspond à la dernière instruction donnée.
Catégorie: instruction
Exemple: `SET PASSWORD "MSX"`

**SET PROMPT <X$>**
Après chaque commande, on voit toujours apparaître 'Ok'. Il s'agit du message guide-opérateur appelé 'prompt' en anglais. Cette instruction permet de remplacer ce mot par la chaîne X$. Voir aussi SET PASSWORD.
Catégorie: instruction
Exemple: `SET PROMPT "READY"`

**SET SCREEN**
Cette instruction permet de stocker une série de paramètres qui seront utilisés à chaque mise en route du système.
Le tableau suivant montre quelles sont les possibilités:

| | |
|---|---|
| Mode SCREEN | 0 ou 1 |
| WIDTH (nombre de colonnes) | 1 à 80 |
| Code couleur écriture | 0 à 15 |
| Code couleur fond | 0 à 15 |
| Code couleur pourtour | 0 à 15 |
| Touches de fonction | ON ou OFF |
| Son en appuyant sur les touches | ON ou OFF |
| Mode imprimante | MSX ou non-MSX |
| Nombre de bauds, cassette | 1200 ou 2400 |
| Page affichée | 0 à 3 |

Catégorie: instruction
Exemple:
```
10 SCREEN 0:WIDTH 80:KEY OFF
20 SET SCREEN:END
```

**SET TIME<X$>[,A]**
Cette instruction permet de régler l'heure gérée par le circuit d'horloge, selon le format suivant HH:MM:SS. En donnant A, HH et MM figurent comme heure d'alarme. L'heure est conservée, même lorsque l'ordinateur est éteint.
Catégorie: instruction
Exemple: `SET TIME "11:55:06"`

**SET TITLE <X$>[,<Y>]**
Cette instruction permet d'afficher lors de la mise en route du système le titre X$ dans la couleur <Y> (voir SET PASSWORD).
Si X$ à une longeur de six caractères ou plus (mais n'affiche que 6 caractères), le système attendra jusqu'a ce qu'une touche soit tapée avant de démarrer.
Catégorie: instruction
Exemple: `SET TITLE "ALLO"`

**SET VIDEO<X>[,<Y>[,<Z>[,<XX>[,<YY>[,<ZZ>[,<XXX>]]]]]]**
Cette instruction permet de passer en mode superimpose. Elle ne peut être utilisée que si l'ordinateur offre cette possibilité. Superimpose signifie que l'on peut mélanger des images (les superposer). Les possibilités sont résumées dans le tableau suivant:

| *<X>* | *D'où vient l'image ?* |
|---|---|
| 0 | de l'ordinateur |
| 1 | de l'ordinateur |
| 2 | avec une autre image (superposée) |
| 3 | de l'entrée vidéo |

En 0, il n'est pas possible de réaliser une synchronisation externe et dans 1, 2 ou 3, il n'y a pas de 'sortie video composite'.

| *<Y>* | *Quelle est l'importance de l'intensité ?* |
|---|---|
| 0 | demi |
| 1 | entière |

Sauf indication contraire, Y est égale à 0.

<Z> *détermine la commande des couleurs*
0 Uniquement pour sortie (exécution)
1 Uniquement pour entrée (saisie)

La valeur par défaut est 0.

<XX> *détermine la synchronisation*
0 Interne
1 Externe

La valeur par défaut est 0.

<YY> *détermine le signal sonore*
0 uniquement de l'ordinateur
1 mélange les données externes au canal de droite de l'ordinateur
2 mélange les données externes au canal de gauche de l'ordinateur
3 mélange les canaux externes avec l'ordinateur

La valeur par défaut est 0.

<ZZ> *détermine l'entrée vidéo externe*
0 Euroconnecteur RVB
1 Connecteur CVBS (PAL ou SECAN)

La valeur par défaut est 0.

<XXX>*Sélection de la sortie audio/vidéo de l'euroconnecteur RGB*
1 est choisi
0 n'est pas choisi

La valeur par défaut est 0.
Catégorie: instruction
Exemple: `SET VIDEO 2`
*Attention:* cette instruction ne peut être utilisée si l'ordinateur n'a pas un mode superimposé

**SGN(<X>)**
Si X > 0; SGN (X) = 1
Si X = 0; SGN (X) = 0
Si X < 0; SGN (X) = -1
Catégorie: fonction
Exemple: `PRINT SGN(31)`

**SIN(<X>)**
Donne le sinus de (X). <X> doit être spécifié en radians.
Catégorie: fonction
Exemple: `PRINT SIN(3.1415/12)`

**SOUND <numéro de registre, valeur>**
Instruction pour générer un son. Les conventions suivantes s'appliquent:

| *registre* | *valeur* | *signification* |
|---|---|---|
| 0 | 0-255 | fréquence canal A |
| 1 | 0-15 | fréquence canal A |
| 2 | 0-255 | fréquence canal B |
| 3 | 0-15 | fréquence canal B |
| 4 | 0-255 | fréquence canal C |
| 5 | 0-15 | fréquence canal C |
| 6 | 0-31 | fréquence souffle |
| 7 | 0-63 | choix de canal: |
| 8 | 0-15 | volume canal A |
| 9 | 0-15 | volume canal B |
| 10 | 0-15 | volume canal C |
| 11 | 0-255 | forme et période d'enveloppe |
| 12 | 0-255 | forme et période d'enveloppe |
| 13 | 0-14 | choix de la forme |

Catégorie: instruction
Exemple: `10 FOR K=1 TO 10:SOUND K,0:NEXT K`

**SPACE$(<X>)**
Donne une chaîne consistant en <X> espaces. <X> est arrondi à un nombre entier et doit se situer entre 0 et 255.
Catégorie: fonction
Exemple: `10 A$=SPACE$(20):PRINT A$;"A"`

**SPC(<X>)**
Affiche <X> espaces sur l'écran ou sur l'imprimante. SPC peut être utilisée seulement dans une instruction PRINT ou LPRINT. <X> doit se situer entre 0 et 255.
Catégorie: fonction
Exemple: `PRINT "MSX";SPC(3);"2"`

**SPRITE ON**
**SPRITE OFF**
**SPRITE STOP**
ON, OFF et STOP indiquent si l'ordinateur fait attention à la 'collision' de sprites. Voir ON SPRITE GOSUB. SPRITE STOP mémorise la collision en attendant SPRITE ON. Dans ce cas, le saut au sous-programme indiqué par ON SPRITE GOSUB sera effectué seulement quand SPRITE ON sera trouvé de nouveau.
Catégorie: instruction
Exemple:

```
10 DATA 60,66,165,129,165,153,66,60
20 DATA 60,126,219,255,255,219,102,60
30 A$=""
40 FOR I=1 TO 8
50 READ A:A$=A$+CHR$(A)
60 NEXT
70 B$=""
80 FOR I=1 TO 8
90 READ A:B$=B$+CHR$(A)
```

```
100 NEXT
110 SCREEN 2,1:COLOR 15,4,1
120 ON SPRITE GOSUB 210
130 SPRITE$(0)=A$:SPRITE$(1)=B$
140 SPRITE ON
150 A=INT(RND(1)*256):B=INT(RND(1)*256)
160 FOR I=0 TO 191
170 PUT SPRITE 0,(A,I),1
180 PUT SPRITE 1,(B,191-I),15
190 NEXT:GOTO 140
200 SPRITE OFF
210 PLAY "L4CDEFEDCREFGAGFER"
220 PUT SPRITE 0,(0,208)
230 PUT SPRITE 1,(0,208)
240 I=191:RETURN
```

**SPRITE$(<numéro>)=<expression chaîne>**
Le <numéro> indique le numéro de sprite. En fonction de la définition du format de sprite tel que donné par l'instruction SCREEN, <numèro> peut être au maximum 63 ou 255. On indique la forme du sprite au moyen de <expression chaîne>. Habituellement, cela se fait au moyen d'un nombre de fonctions CHR$, par exemple:

```
SPRITE$(1) = CHR$(&H18) + CHR$(&H3C) + CHR$(&HFF) + CHR$(&H99) + CHR$(&H99)
             + CHR$(&HFF) + CHR$(&HC3) + CHR$(&HFF)
```

Pour une sprite 8x8 on a ainsi besoin de 8 fonctions CHR$
SPRITE$ peut être utilisé dans les différents modes graphiques (voir aussi PUT SPRITE).
Pour colorier les sprites, voir COLOR SPRITE et COLOR SPRITE$.
Catégorie: instruction
Exemple: voir SPRITE ON/OFF/STOP

**SQR(<X>)**
Donne la racine carrée de (X). <X> doit être supérieur ou égal à zéro.
Catégorie: fonction
Exemple: `PRINT SQR(4)`

**STICK(<X>)**
Donne la position de la manette. Si l'on donne à X la valeur 0, les touches du curseur sont prises en considération.
1 concerne la manette No 1
2 la manette No 2
Les valeurs correspondent aux directions suivantes.
Catégorie: fonction
Exemple: `PRINT STICK(0)`

**STOP**
Interrompt l'exécution d'un programme. On peut reprendre l'exécution du programme au moyen de la commande CONT.
STOP peut être placé n'importe où dans le programme.
Catégorie: instruction

Exemple:
```
10 INPUT A
20 PRINT A:IF A=0 THEN STOP
30 GOTO 10
```

**STOP ON**
**STOP OFF**
**STOP STOP**

Règle le mode d'une interruption causée par CTRL/STOP (voir ON STOP GOSUB). A STOP, la situation d'interruption est maintenue, c'est-à-dire que le saut au sous-program me trouvé par ON STOP GOSUB est effectué seulement quand STOP ON est trouvée d nouveau.
Catégorie: instruction
Exemple:
```
10 ON STOP GOSUB 50
20 STOP ON
30 INPUT A$
40 IF A$="END" THEN STOP OFF:END ELSE GOTO 30
50 PRINT "TOUCHE END (+RETURN)":RETURN
```

**STRIG (<X>)**

Indique si le bouton d'une manette à été enfoncé. (–1 = enfoncé, 0 = non enfoncé)
*<X> signification*
**0** barre espace du clavier
**1** touche 1 de la manette 1.
**2** touche 1 de la manette 2.
**3** touche 2 de la manette 1.
**4** touche 2 de la manette 2.
Catégorie: fonction
Exemple: `PRINT STRIG(0)`

**STRIG (<x>) ON**
**STRIG (<x>) OFF**
**STRIG (<x>) STOP**

Règle le mode d'une interruption causée par la manette ou par la barre espace (voir ON STRIG GOSUB). STRIG STOP suspend la validation et mémorise l'action sùr la barr espace ou sur les boutons de validation en attendant STRIG ON.
Catégorie: instruction
Exemple:
```
10 CLS:ON STRIG GOSUB 40
20 STRIG(0) ON
30 GOTO 30
40 LOCATE 5,5:PRINT "BARRE D'ESPACEMENT ENFONCEE"
50 FOR I=1 TO 300:NEXT:LOCATE 5,5:PRINT SPC(27)
60 RETURN
```

**STR$ (<X>)**

Transforme un nombre en chaîne de caractères.
Catégorie: fonction
Exemple: `10 A$=STR$(10):PRINT A$`

**STRING$(<I>,<J>) ou STRING$(<I>,<X$>)**

Donne une chaîne consistant en <I> caractères identiques qui ont <J> comme code

ASCII ou qui correspondent au premier caractère de <X$>.
Catégorie: fonction
Exemple: `PRINT STRING$(4,65)`

**SWAP <variable>,<variable>**
Echange les valeurs de deux variables.
Ces variables doivent toutefois être du même type.
Catégorie: instruction
Exemple: `10 A=3:B=5:SWAP A,B:PRINT A,B`

**TAB(<X>)**
place le curseur à la position <X> de la ligne. Si le curseur est déjà au-delà de la position <X>, rien ne se passe.
<X> doit se situer entre 0 et 255. 0 est la position la plus à gauche. TAB ne peut être utilisée qu'avec une instruction PRINT ou LPRINT.
Catégorie: fonction
Exemple: `PRINT TAB(12);"*"`

**TAN(<X>)**
Donne la tangente de <X>. <X> doit être spécifié en radians.
Catégorie: fonction
Exemple: `PRINT TAN(3,1415/8)`

**TIME**
La variable système TIME est augmentée de 50 fois par seconde. On peut utiliser cette variable pour obtenir de vrais nombres aléatoires au moyen d'une fonction RND (voir cette fonction).
Catégorie: variable de système
Exemple: `PRINT INT(RND(-TIME)*6+1)`

**TROFF**
**TRON**
TRON permet de visualiser les numéros d'instructions dans la partie suspecte du programme.
Ceci facilite la recherche d'erreurs. Cette commande est annulée par TROFF.
Catégorie: commande
Exemple:
```
10 FOR I=1 TO 3
20 PRINT I
30 NEXT
40 END
TRON
Ok
RUN
[10][20]1
[30][20]2
[30][20]3
[30][40]
Ok
TROFF
Ok
```

**USR[<n>](<X>)**
Indique un programme en langage machine. <n> est le nombre attribué au programme en langage machine par DEF USR. Si <n> est omis, USR0 est pris par défaut.
<X> est un argument qui est passé au programme en langage machine.
L'argument <X> est passé de la manière suivante:

1. valeurs chaîne: l'adresse &HF663 contient la valeur 3. Les adresses &HF7F8 et &HF7F9 indiquent au moyen d'un adressage indirect ou se trouve cette valeur. La valeur est stockée dans trois octets: l'octet 1 contient la longueur, et les octets 2 et 3 contiennent l'adresse mémoire de la valeur alphanumérique.
2. Nombres entiers: l'adresse &HF663 contient la valeur 2. Les adresses &HF7F8 et &HF7F9 contiennent la valeur entière.
3. Valeurs en simple précision: l'adresse &HF663 contient la valeur 4.

Les adresses &HF7F6 à &HF7F9 contiennent la valeur en simple précision.

4. Double précision : l'adresse &HF663 contient la valeur 8. Les adresses &HF7F6 à &HF7FD contiennent la valeur en double précision.

Les valeurs qui doivent être renvoyées au MSX BASIC à partir du sous-programme doivent être stockées aux endroits susmentionnés par le sous-programme en langage machine.
CLEAR permet de réserver l'espace nécessaire.
Catégorie: fonction
Exemple:

```
 10 CLEAR 200,&HEFFF
 20 AB=&HF000
 30 FOR I=AB TO AB+9
 40 READ A$:A=VAL("&H"+A$)
 50 POKE I,A
 60 NEXT I
 70 DEFUSR=&HF000
 80 INPUT"INTRODUIRE UN NOMBRE ENTIER";A%
 90 PRINT "NOMBRE ENTIER =";A%
100 R=USR(A%)
110 PRINT "RESULTAT EST UN NOMBRE ENTIER PLUS 1";R
120 END
130 DATA 23,23,4E,23,46,03,70,2B,71,C9
```

**VAL(<X$>)**
Indique la valeur numérique de <X$>. Quand le premier caractère de <X$> n'est pas +, -, & ou un chiffre, VAL devient égal à 0.
Catégorie: fonction
Exemple: `PRINT VAL("10")`

**VARPTR(<variable>)**
**VARPTR(#<X>)**
Donne l'adresse du premier octet de la valeur qui appartient à la <variable>, ou le premier octet d'un bloc de commande fichier.
Avant que la fonction ne puisse être appelée, il faut attribuer au préalable une valeur à <variable> ou à <X>.
Catégorie: fonction

Exemple:
```
10 A=10:B=VARPTR(A)
20 IF B<0 THEN B=B+65536
30 C$="0000"+HEX$(B)
40 PRINT RIGHT$(C$,4):END
```

**VDP(<X>)**
Concerne le contenu des registres VDP. X doit être compris entre 0 et 24 ou entre 33 et 47.
Le registre 8 ne peut être que lu.
Catégorie: variable de système
Exemple:
```
10 FOR I=0 TO 8
20 A=VDP(I):B$="00000000"+BIN$(A)
30 PRINT RIGHT$(B$,8)
40 NEXT
```

**VPEEK(<X>)**
Donne le contenu de l'adresse <X> de la mémoire vidéo. X doit être compris entre 0 et 65535. Dans les modes écran 5 à 8, l'adresse absolue est égale à X + l'adresse de départ de la page active. Celle-ci découle du tableau suivant:

| *mode écran* | *Calcul* |
|---|---|
| Mode écran 5 | numéro de page × &H08000 |
| Mode écran 6 | numéro de page × &H08000 |
| Mode écran 7 | numéro de page × &H10000 |
| Mode écran 8 | numéro de page × &H10000 |

Catégorie: fonction
Exemple:
```
10 A=VPEEK(0)
20 A$="00"+HEX$(A)
30 PRINT RIGHT$(A$,2)
```

**VPOKE <X, donnée>**
Identique à VPEEK, mais pour écrire dans la mémoire Video (VRAM).
Catégorie: instruction
Exemple:
```
10 VPOKE(0),(VPEEK(0)):END
```

**WAIT <numéro de port,octet 1>[,<octet 2>]**
Concerne l'introduction de données par l'entrée port <numéro de port> indiqué.
La valeur introduite est combinée avec <octet 1> via la condition OR exclusive. Ce résultat est combiné avec <octet 2> par une opération AND. Si ce resultat est 0, l'entrée est poursuivie. Sinon, l'ordinateur continue le programme à l'instruction suivante. Si <octet 2> est omis, la valeur 0 est prise par défaut.
Catégorie: instruction
Exemple:
```
10 WAIT &HA8,240
```

**WIDTH (<nombre de colonnes>)**
Fixe le nombre de colonnes de l'écran texte.
Dans le mode SCREEN 0 on peut choisir entre 1 à 80 colonnes. Dans le mode SCREEN 1, on peut choisir entre 1 à 32 colonnes (voir SCREEN).
Avec certains téléviseurs, la première colonne n'apparaît pas. SET ADJUST vous permet de régler l'image.

Signalons encore que lorsque l'ordinateur est mis en route, l'image est réglée selon le mode SCREEN 0 et compte 37 colonnes (à moins que l'on n'utilise SET SCREEN). Si l'on passe au mode SCREEN 1, l'image comptera 29 colonnes.
Catégorie: instruction
Exemple: `SCREEN 0:WIDTH 80`

# MSX-DOS

*Cette partie du manuel vous est utile si votre micro ordinateur Philips-MSX est fourni avec le système d'exploitation MSX- DOS.*

# 1

## L'AIDE À L'UTILISATEUR MSX-DOS PHILIPS

**AVERTISSEMENT**

Par précaution, il faut copier cette disquette avant d'utiliser USER SHELL. Avant de faire la copie, vous devez mettre USER SHELL dans la position d'interdiction d'écriture. Ainsi vous évitez l'écriture accidentelle sur la disquette originale.

Vous vous êtes procuré une disquette MSX-DOS PHILIPS, un système d'exploitation qui apportera de nombreux avantages supplémentaires à votre ordinateur MSX PHILIPS.

**Qu'est-ce qu'une aide à l'utilisateur**

Une aide à l'utilisateur est, comme son nom l'indique, une aide pour vous, utilisateur de l'ordinateur MSX PHILIPS. Avec elle, vous apprendrez facilement à utiliser MSX-DOS. Cette aide PHILIPS est même si facile que vous pouvez, sans vraiment utiliser MSX-DOS, en essayer presque toutes les fonctions.

PHILIPS USER SHELL (le nom anglais de cette aide à l'utilisateur) se suffit à lui-même, c'est-à-dire que même sans l'explication de cette introduction, vous pouvez démarrer et l'utiliser.

**Comment démarrer l'aide à l'utilisateur PHILIPS?**

Avant de mettre la disquette PHILIPS dans l'unité de disquettes, vérifiez si tous les appareils sont correctement branchés. Allumez ensuite le téléviseur ou le moniteur, l'unité de disquettes, éventuellement l'imprimante et enfin l'ordinateur. Maintenant on vous demande la date. Pour sauter cette question; appuyez sur RETURN et mettez PHILIPS USER SHELL dans l'unité A.

Pour mettre en marche la disquette PHILIPS, appuyez sur la touche RESET à l'arrière de l'ordinateur. Le programme USER SHELL se charge automatiquement. Pendant ce temps vous voyez affichés sur l'écran des messages nouveaux pour vous. Vous voyez également DOSHLP après 'A>': le curseur MSX-DOS, à propos duquel vous pourrez en apprendre davantage au chapitre 5. DOSHLP est le nom du programme assuré par USER SHELL. Dans MSX-DOS un tel programme est appelé 'fichier'. Nous en reparlerons également.

Si tout est correct, vous verrez le menu principal s'afficher. Il comprend 12 options. En haut de l'écran vous voyez cinq cases foncées (noires si vous avez un téléviseur ou un moniteur couleur), qui représentent les touches de fonction. Lorsqu'une ou plusieurs touches de fonction sont actives, leur possibilité est indiquée dans la case correspondante.

Dans le menu principal seule F5 est disponible, pour appeler les pages 'aide' qui vous expliquent comment utiliser PHILIPS USER SHELL.

En appuyant sur la touche de fonction F5, vous avez un aperçu des différentes fonctions que PHILIPS USER SHELL peut exécuter. On vous explique également comment choisir une rubrique dans le menu. Vous voyez aussi un texte inscrit dans les deux cases de gauche. Vous pouvez quand vous voulez appuyer sur les touches de fonction correspondantes pour faire exécuter par l'ordinateur l'instruction mentionnée dans la case.

**Comment passer de MSX-DOS à l'aide à l'utilisateur?**

En choisissant le numéro 2 du menu principal, vous entrez dans MSX-DOS; le curseur MSX-DOS spécial s'affiche alors. Le système d'exploitation est maintenant en fonctionnement et prêt à recevoir les instructions de son utilisateur. Ce manuel vous expliquera toutes les commandes et vous montrera un certain nombre de fichiers de commandes pratiques.

Si vous voulez revenir de MSX-DOS à PHILIPS USER SHELL, vous le pouvez de deux manières: vous pouvez soit mettre en marche USER SHELL en tapant la commande DOSHLP, puis en appuyant sur RETURN. (c'est le procédé le plus utilisé), soit appuyer sur la touche RESET à l'arrière de l'ordinateur, ce qui a pour résultat de faire redémarrer USER SHELL.

**Comment passer de MSX-Disk BASIC à l'aide à l'utilisateur?**

Si vous êtes allé du menu principal à Disk BASIC par le numéro de sélection 3, vous ne pouvez revenir à USER SHELL qu'en passant par MSX-DOS; ou bien en appuyant sur la touche RESET.

Pour aller de Disk BASIC à MSX-DOS, vous devez taper:

```
CALL SYSTEM
```

Puis appuyer sur la touche RETURN. Après quelques instants le curseur MSX-DOS spécial apparaît sur l'écran;

```
A>
```

Maintenant suivez le même procédé qu'en 3. Tapez DOSHLP et appuyez sur RETURN. Après quelques instants vous voyez à nouveau affiché le menu principal.

**Noms de fichiers dans l'aide à l'utilisateur**

Dans l'aide à l'utilisateur PHILIPS, vous pouvez également utiliser des caractères de substitution (voir chapitre 3) pour entrer les noms de fichier.

L'utilisation des caractères de substitution dans l'aide à l'utilisateur PHILIPS diffère un peu de celle en MSX-DOS.

L'utilisation de l'astérisque comme caractère de substitution dans l'aide à l'utilisateur PHILIPS s'utilise comme suit:

* signifie tous les fichiers sans extension du nom de fichier.

*. suivi d'une extension du nom de fichier signifie tous les fichiers avec l'extension spécifiée.
Un nom de fichier suivi de .* signifie tous les fichiers avec le nom spécifié, sans tenir compte de l'extension.

*.* signifie tous les fichiers.
Un ou plusieurs caractères suivis de * signifie tous les fichiers dont le premier caractère et les caractères suivants sont identiques au caractères spécifiés. Par exemple: AC* signifie tous les fichiers avec un nom commençant par AC.
Si ** est entré comme nom de fichier, l'aide à l'utilisateur PHILIPS suppose qu'on veut entrer *.*.
Si *** est entré comme nom de fichier dans la fonction de duplication, tous les fichiers seront copiés. Une confirmation est demandé pour chaque fichier dans tous les autres cas.

# 2 QU'EST-CE QU'UN SYSTEME D'EXPLOITATION?

**Le système d'exploitation**

Le système d'exploitation est l'aide tacite entre l'ordinateur et son utilisateur. Le système d'exploitation assure votre communication avec les appareils et les programmes. Le système d'exploitation vous permet de communiquer directement avec l'ordinateur, l'imprimante, les unités de disquettes et les autres appareils branchés.

Vous avez besoin d'un système d'exploitation pour travailler avec ces appareils. Vous pouvez le comparer à l'alimentation en courant dans votre maison. Vous avez besoin de ce courant pour faire fonctionner un grille-pain (ou un ordinateur), bien que vous n'y pensiez pas toujours et trouviez ça évident.

Le système d'exploitation est un programme système entièrement adapté à l'ordinateur. Ainsi MSX-BASIC est un système d'exploitation (appelé langage de programmation), qui permet d'écrire des programmes en BASIC et également de commander les appareils branchés, par exemple le magnétophone pour enregistrer les programmes. L'imprimante peut être commandée aussi à l'aide de MSX-BASIC, ainsi que, évidemment, l'unité de disquettes. Le système d'exploitation est spécifique du microprocesseur (ordinateur) utilisé.

Un système d'exploitation est essentiellement une liste de mots stockée dans l'ordinateur, le plus souvent en ROM (la mémoire morte de l'ordinateur). Après l'introduction d'une commande (mot) l'ordinateur cherche dans sa liste de mots 'explicatifs' le sens de ce mot et le convertit dans les instructions qu'il y trouve. Si l'ordinateur ne peut pas trouver le mot (commande), un message d'erreur sera affiché et vous pourrez essayer de nouveau.

Si une unité de disquettes a été branchée à l'ordinateur, on peut utiliser un Disk Operating System (DOS - système d'exploitation à disques). Ainsi MSX-DOS fonctionnera sur un ordinateur MSX et non sur un autre, à moins qu'on change une grande partie des instructions.

MSX-DOS est un système d'exploitation qui permet, au moyen de certaines commandes, d'écrire successivement des données sur une disquette. C'est ce qu'on appelle un fichier. Il permet aussi de corriger ces fichiers et de commander des appareils périphériques (par exemple des imprimantes et d'autres unités de disquettes) s'ils sont branchés à l'ordinateur.

Habituellement la liste des mots explicatifs ne se trouve pas dans la mémoire ROM, mais sur une disquette, et doit donc être chargée dans l'ordinateur avant usage. A cet effet l'ordinateur réservera une partie spéciale de sa mémoire pour y ranger la liste de mots. Si l'on quitte DOS, l'espace réservé sera perdu et sera disponible pour d'autres fonctions. Pour revenir à DOS, la liste

devra être à nouveau chargée au moyen d'une commande spéciale prévue à cet effet.

MSX-DOS deviendra de plus en plus important pour l'utilisateur, car un grand nombre de programmes commercialisés fonctionneront sous MSX-DOS. Les principaux programmes livrables (d'ici peu) seront PASCAL, un interpréteur BASIC, MULTIPLAN, etc.

**Travailler avec un système d'exploitation**

Le système d'exploitation MSX-DOS comprend un certain nombre de commandes spécifiques qui ne sont pas utilisables avec MSX-Disk BASIC, bien que certaines d'entre elles soient analogues. Pour éviter la confusion, les différences seront expliquées au chapitre 4. Le système d'exploitation à disques est toujours lu et stocké de la disquette dans un espace de mémoire réservé spécialement pour le DOS. Si la capacité de mémoire est insuffisante, une telle réservation est impossible. Pour cette raison, MSX-DOS ne fonctionne que sur des ordinateurs MSX ayant une capacité de mémoire de 64 K RAM ou plus (la RAM est également appelée la mémoire vive).

Comme il a été expliqué ci-dessus, le système d'exploitation ne peut être chargé qu'à partir de la disquette. Si le système d'exploitation est démarré 'à froid', tous les appareils sont arrêtés. Il faut d'abord mettre en marches les appareils périphériques et enfin l'ordinateur; puis la disquette système MSX -DOS doit immédiatement être insérée dans l'unité A.

Si l'ordinateur est déjà en service en DISK-BASIC, on peut aussi utiliser le démarrage 'à chaud'. Pour cela, il faut insérer la disquette de systèmes MSX-DOS dans l'unité A et appuyer sur le bouton RESET à l'arrière de l'ordinateur.

Comme l'ordinateur reconnaît l'interface de l'unité de disques, il mettra en marche l'unité de disquettes lorsque le numéro de la version du système MSX aura été entré. Le fichier MSX-DOS est alors recherché qui doit assurer la lecture des différentes commandes à partir du fichier de commandes. Le système lui-même initialisera les paramètres requis. La plupart de ceux-ci ne sont pas intéressants pour l'utilisateur et ne servent qu'à indiquer à l'ordinateur qu'une unité de disquettes est branchée et laquelle. Ceci est très important, parce que les disquettes sont divisées différemment.

**Fichiers MSX-DOS**

Votre disquette système MSX-DOS comprend 4 fichiers: COMMAND.COM, MSXDOS.SYS, DOSHLP.COM et AUTOEXEC.BAT. Vous verrez au chapitre suivant que les fichiers finissant par .COM sont des fichiers de commandes en langage machine; le fichier MSXDOS.SYS assure le chargement des commandes qui sont stockées en langage machine, dans le fichier COMMAND.COM. Il est sage de copier la disquette systè-

me immédiatement après l'achat de MSX-DOS. Il y a donc au moins 4 fichiers sur la disquette système, mais il se peut que votre disquette système en comporte plus ou moins. Dans ce cas il est recommandable de copier d'abord la disquette entière après le démarrage.

Avant le démarrage de MSX-DOS, il est recommandé de consulter le chapitre 5 pour procéder correctement au démarrage et copier la disquette système. Ce chapitre vous expliquera aussi comment préparer une disquette neuve pour copier la disquette système.

Votre disquette peut contenir toutes sortes de fichiers comme des fichiers de programmes normaux, des fichiers de langage machine, des fichiers ASCII, des fichiers de textes, etc. Tous ces fichiers devant être reconnaissables pour vous, leur utilisateur, le nom de fichier est prolongé (on parle d'extension). Ce prolongement vous permet de savoir de quelle sorte de fichier il s'agit. Les différentes extensions admises et interdites seront étudiées au chapitre suivant. Pour le moment, il suffit que vous sachiez que le fichier COMMAND a l'extension COM, de fichier de COMmandes, et le fichier MSXDOS a l'extension SYS, de SYStème. Enfin, le fichier DOSHLP.COM est le fichier de commandes pour l'aide à l'utilisateur, et le fichier AUTOEXEC.BAT assure que la disquette système démarre toujours avec l'aide à l'utilisateur.

# 3 DIFFÉRENCES ENTRE MSX-DOS ET MSX-DISK BASIC

**Qu'est-ce que MSX-Disk BASIC?**

Ce chapitre a été écrit spécialement pour ceux qui travaillent déjà depuis un certain temps avec MSX-Disk BASIC et désirent maintenant passer à MSX-DOS. Nous énumérerons un certain nombre de différences importantes auxquelles il faut être bien attentif. A cause de ces différences dans les commandes il vous arrivera d'utiliser accidentellement en MSX BASIC une commande utilisée en Disk BASIC.

Heureusement MSX-DOS vous signalera que c'est une fausse commande, mais bien sûr il est préférable de connaître les différences et de donner en une fois la commande correcte pour la fonction désirée.

**Analogie entre les commandes**

Heureusement un certain nombre de commandes restent les mêmes en Disk BASIC et en MSX-DOS. Celles-ci ne poseront aucun problème. Le nom des fichiers de programmes et de données est parmi les choses les plus importantes qui ne doivent pas changer. Il doit être bien sûr possible d'appeler à partir de MSX-DOS les fichiers créés en Disk BASIC. La structure du nom de fichier est par conséquent identique, ainsi que le nombre de caractères, l'extension, les caractères utilisés, l'extension recommandée, etc.

Dans la recherche des différents fichiers, les mêmes caractères de substitution sont admis. Les noms de fichier réservés en Disk BASIC valent aussi en MSX-DOS.

En fait il n'y a que deux commandes Disk BASIC reconnues aussi en MSX-DOS, à savoir FORMAT et COPY. En Disk BASIC la commande FORMAT doit être appelée par CALL ou en abrégé par '–'. En MSX-DOS cet appel n'est plus nécessaire et seule la commande suffit pour formater une disquette neuve.

La commande COPY (malgré l'analogie) posera certainement des problèmes, surtout au début. En Disk BASIC on doit vraiment copier de l'une 'à' (anglais: TO) l'autre. En MSX-DOS ce n'est plus nécessaire, parce que le système sait que la commande COPY est suivie de deux noms: le premier nom de fichier est le nom du fichier original et le deuxième nom de fichier en sera la copie. De même, les guillemets nécessaires en MSX BASIC sont refusés en MSX-DOS ce qui amène le message:

```
File not found
```

En MSX BASIC on peut entrer successivement, sans signe de séparation, la commande, le premier nom de fichier, TO et le deuxième nom de fichier. En principe les guillemets sont déjà considérés comme signe de séparation entre les différents noms

et commandes. Avec MSX-DOS il faut absolument utiliser l'<ESPACE> comme signe de séparation (voir paragraphe 6.2 pour les autres signes de séparation); sans les signes de séparation spéciaux, on reçoit irrévocablement un signal d'erreur. Les guillemets ne doivent plus être utilisés.

**Différences entre les commandes**

Le passage aux nouvelles commandes en MSX-DOS cause probablement le plus de problèmes, bien que cela dépende naturellement du temps qui a été consacré à travailler avec Disk BASIC.

Une des premières choses qu'on consulte est le répertoire qu'on appelle normalement avec la commande 'FILES'. En MSX-DOS on utilise l'abréviation DIR pour DIRectory. Toutes les variantes existantes de la commande 'FILES' utilisables en Disk BASIC, sont aussi utilisables en MSX-DOS, à l'exception des guillemets comme signe de séparation entre la commande et le nom de la disquette, de l'unité de disquettes ou le nom de fichier.

L'effacement d'un fichier sur la disquette se fait à l'aide de la commande 'KILL' en Disk BASIC. MSX-DOS connaît deux commandes pour cela, à savoir DEL et ERASE. Pour le reste, toutes les règles concernant les guillemets et les signes de séparation sont également valables.

Le changement d'un nom de fichier se fait en Disk BASIC au moyen de la commande NAME "<nom de fichier>" AS "<nom de fichier>". En MSX-DOS cette commande est REN (de REName) et elle est donnée avec des espaces comme signes de séparation.

Soyez particulièrement attentif à la commande CALL SYSTEM: elle ne fonctionne qu'en MSX-Disk BASIC si vous avez démarré avec MSX-DOS. Cette commande permet de revenir en MSX-DOS si vous en êtes sorti par la commande BASIC.

Si vous n'avez pas démarré en MSX-DOS, l'ordinateur ne reconnaîtra pas la commande. Cela signifie que MSX-DOS ajoute lors du démarrage une commande à 'la liste de mots de' MSX-Disk BASIC.

Si vous voulez toutefois démarrer MSX-DOS sans la commande CALL SYSTEM, vous pouvez utiliser le démarrage à chaud, qui sera étudié au chapitre suivant.

**Résumé**

En MSX-Disk BASIC tous les noms de fichier doivent être indiqués entre guillemets (quotes), pas en MSX-DOS.

Ci-dessous, les commandes pour la même fonction en DISK-BASIC et en MSX-DOS.

| *MSX-Disk BASIC* | *MSX-DOS* |
|---|---|
| CALL FORMAT | FORMAT |
| FILES | DIR |
| KILL | DEL ou ERASE |
| NAME AS | REN ou RENAME |
| COPY TO | COPY |

# 4 COMMENT COMMENCER AVEC MSX-DOS

**Démarrage**

Le démarrage du système d'exploitation MSX-DOS peut se faire de deux façons: le démarrage 'à froid' et le démarrage 'à chaud'. Dans le cas du démarrage 'à froid' l'ordinateur et les appareils périphériques (téléviseur, imprimantes, unités de disquettes) sont arrêtés. Vous allumez d'abord les appareils périphériques et enfin l'ordinateur. Puis vous insérez immédiatement la disquette système dans l'unité de disquettes.

Avec le démarrage 'à chaud', l'ordinateur et les appareils périphériques sont en marche. Vous devez insérer d'abord la disquette système dans l'unité de disques et puis appuyer sur le bouton RESET de l'ordinateur.

S'il y a plusieurs unités de disquettes, la disquette doit être insérée dans l'unité A. S'il n'y en a qu'une, celle-ci est automatiquement l'unité A ou, autrement dit, l'unité par défaut. Après la mise en marche de l'unité de disquettes et de l'ordinateur (ou lorsqu'on appuie sur le bouton RESET), l'ordinateur donnera après quelques instants le message suivant:

```
MSX-DOS version 1....
Copyright 1984 by Microsoft

Command version 1....
```

**Le curseur**

Après le démarrage du système, le curseur apparaît:

```
A>
```

La lettre indique à partir de quelle unité de disquettes le démarrage s'est fait. C'est toujours à partir de l'unité A, appelée aussi l'unité par défaut. Le démarrage de MSX-DOS doit toujours se faire à partir de l'unité A.

Si le fichier MSXDOS.SYS ne se trouve pas sur la disquette dans l'unité A, le système démarrera de lui-même en Disk BASIC. S'il s'y trouve, on peut sortir de Disk BASIC en arrêtant l'ordinateur, en insérant la disquette système MSX-DOS dans l'unité de disquettes A et en remettant en marche l'ordinateur (démarrage à froid). On peut aussi insérer d'abord la disquette système dans l'unité A et appuyer sur le bouton RESET à l'arrière de l'ordinateur (démarrage à chaud). Si l'on a démarré en MSX-DOS, on peut, au moyen d'une commande, passer en Disk BASIC et revenir au système d'exploitation.

Le curseur indique donc quelle unité de disquettes est en service. Pour changer d'unité il faut indiquer après le curseur la lettre de l'autre unité de disquettes qu'on désire utiliser, suivie de deux points.

```
A>B:
```

Par cette commande le curseur se changera en:

```
B>
```

et l'unité B sera l'unité par défaut. Tant que l'unité de disquettes n'est pas active, vous ne verrez rien de ce changement de curseur. Ce n'est que lorsqu'une commande pour l'unité de disquettes sera entrée, que le système attirera votre attention sur le fait qu'il faut insérer une autre disquette (dans le cas d'une seule unité).

L'unité de disquettes B restera l'unité par défaut jusqu'à ce qu'une autre unité soit nommée à sa place.

**Unité de disquettes par défaut**

Lors du démarrage l'unité de disquettes A est donc toujours l'unité par défaut. Le choix d'une autre unité de disquettes après le démarrage se fait par la commande suivante:

```
B: ou C: ou D:
```

Ces lettres peuvent êtres des majuscules ou des minuscules; le système convertit toutes les lettres de commandes en majuscules. La commande

```
b: ou c: ou d:
```

est donc aussi correcte et équivaudra pour le système à la commande en majuscules.

S'il y a plus d'une unité, vous avez le choix entre l'unité A et l'unité B, mais s'il n'y a pas de deuxième unité branchée, le système demandera une autre disquette dès que l'unité B sera en marche.

L'unité par défaut est A et par la commande B: vous passez au curseur B qui apparaîtra sur l'écran; il ne se passe rien d'autre pour le moment. Dès que vous demandez le répertoire ou que vous entrez une autre commande, le système donne le message suivant:

```
Insert diskette for drive B:
and strike a key when ready
```

Après le changement de disquette vous appuyez sur une touche et la commande déjà entrée est exécutée. La disquette B reste maintenant la disquette par défaut jusqu'à ce que vous rappeliez un autre curseur et donniez une commande pour la disquette.

Après l'entrée d'une combinaison impossible le système donnera le message:

```
Invalid drive specification
A>
```

S'il n'y a qu'une seule interface, les commandes C: ou D: donneront ce message d'erreur.

**Répertoire**

Toutes les disquettes ont un espace réservé aux données des différents fichiers se trouvant sur la disquette. Naturellement il est important pour l'utilisateur de voir rapidement quels fichiers se trouvent sur la disquette.
Pour cela MSX-DOS possède la commande:

```
DIR
```

de DIRectory (répertoire). Lorsque vous avez démarré l'ordinateur en MSX-DOS et que l'écran affiche le curseur, l'ordinateur attend une commande, par exemple DIR. L'écran affiche alors:

```
A>dir ou A>DIR
```

Il en résulte, lorsqu'on appuie sur <RETURN>, l'énumération de tous les fichiers se trouvant sur la disquette, avec les données les plus importantes telles que la longueur en octets et la dernière date de mise à jour du fichier. Après le démarrage de votre disquette système et la commande DIR, les fichiers suivants seront toujours dans le répertoire:

```
MSXDOS     SYS      2560  1-01-84
COMMAND    COM      6528  1-01-84
DOSHLP     COM     32768  1-01-84
AUTOEXEC   BAT         9  1-01-84
         4 files           318464 bytes free
```

La longueur en octets par fichier peut différer en fonction de la version de la disquette système. Les deux premiers fichiers sont pour MSX-DOS et lui sont indispensables.

Le premier fichier est le fichier système MSX-DOS et le deuxième est le fichier de commandes comportant les commandes MSX-DOS. Vous pourrez en savoir davantage sur la commande DIR au chapitre 6.

**Formatage**

La commande FORMAT est une commande importante pour préparer les disquettes. Cette commande assure la préparation de la disquette à recevoir des fichiers MSX-DOS. Toute disquette neuve doit donc être formatée avant son utilisation. Des disquettes déjà formatées en MSX Disk BASIC conviennent pour MSX DOS et il n'est donc pas nécessaire de les formater à nouveau. Ainsi vous pouvez ajouter les deux fichiers MSX-DOS à vos disquettes déjà munies de programmes. Dès que le curseur est affiché, vous entrez la commande pour le formatage:

```
A>FORMAT
```

Suivez les instructions à l'écran et attendez jusqu'à ce que le message suivant apparaisse:

```
Format complete
```

Il est sage de préparer, donc de formater, d'abord un certain

nombre de disquettes. Copiez ensuite sur les disquettes formatées les deux fichiers pour le démarrage de MSX-DOS.

**Copies de sécurité**

Il est recommandé de faire une copie de sécurité de toutes vos disquettes. Si une disquette est endommagée ou un fichier détérioré, vous serez heureux d'avoir une disquette de sécurité. Cela s'applique naturellement à la disquette système MSX-DOS. Il est conseillé de copier les fichiers MSX-DOS sur toutes vos disquettes, afin de démarrer et utiliser immédiatement le système d'exploitation. Une copie complète de la disquette système est recommandable. Travaillez ensuite seulement avec cette copie de sécurité et gardez la disquette système originale à part. Pour faire une copie, MSX-DOS possède la commande:

```
COPY
```

Cette commande transfère un fichier d'une disquette sur une autre.
Avec la commande

```
COPY *.* B:
```

tous les fichiers de la disquette système sont maintenant transférés sur la disquette B, qui servira maintenant de disquette de travail. Vous pouvez aussi copier seulement les fichiers COMMAND.COM et MSXDOS.SYS sur une autre disquette avec la commande COPY.

Vous pouvez maintenant transférer tous les fichiers MSX-DOS sur une disquette déjà pourvue de vos propres programmes ou sur des disquettes neuves que vous venez de formater.

Si vous avez démarré avec l'unité de disquettes A et si vous désirez copier seulement les deux fichiers MSX-DOS sur une deuxième disquette, il faut entrer la commande suivante:

```
COPY MSXDOS.SYS B:
```

Faites bien attention aux espaces entre les différentes commandes. Tout d'abord nous demandons à l'ordinateur de copier (COPY), après un espace nous entrons une deuxième commande qui comporte le nom du fichier à copier (MSXDOS.SYS), puis après un autre espace nous donnons la dernière commande qui indique où la copie doit se faire (B:), et nous appuyons sur la touche <RETURN>. L'ordinateur lira le fichier MSXDOS.SYS de l'unité ou de la disquette par défaut et ensuite il demandera à l'unité de disquettes (dans le cas d'une seule unité de disquettes) d'inscrire le fichier sur la disquette B. Le fichier écrit, l'ordinateur signalera:

```
1 file copied
```

Il faut maintenant copier le fichier suivant avec la commande:

```
COPY COMMAND.COM B:
```

La copie faite, l'ordinateur le signalera et nous aurons une copie des fichiers MSX-DOS.

On peut également copier la disquette entière avec une seule commande. La commande COPY a encore d'autres possibilités qu'on étudiera au chapitre 6.

La duplication rapide des fichiers sur plusieurs disquettes sans le recours à la commande COPY est possible en utilisant les 'touches d'édition' qu'on étudiera au chapitre 8.

# 5 COMMANDES MSX-DOS

**Qu'est-ce qu'une commande?**

La communication entre l'ordinateur et tous les appareils qui lui sont branchés se fait au moyen de commandes. En tapant une commande MSX-DOS sur le clavier, vous pouvez demander (donner la commande) au système d'exécuter des tâches importantes et utiles. Ainsi il y a des commandes pour copier, afficher, effacer un fichier ou en changer le nom. Il y a aussi des commandes pour formater une disquette, exécuter un programme, afficher tous les fichiers sur la disquette ou changer la date et l'heure. Vous pouvez également régler l'écran et l'imprimante et enfin combiner plusieurs commandes en un seul fichier de commandes.

Nous avons trois possibilités pour faire exécuter les commandes, à savoir en entrant des

commandes MSX-DOS
noms de fichier avec l'extension .COM
nom de fichier avec l'extension .BAT

Les commandes les plus simples et les plus habituelles sont les commandes spécifiques MSX-DOS qui sont invisibles, parce qu'elles sont incluses dans le fichier 'COMMAND.COM', comme vous avez pu le lire au chapitre 2. Quand vous entrez une telle commande et que vous appuyez sur la touche <RETURN>, cette commande est exécutée immédiatement.

Les autres commandes se trouvent sur disquette sous la forme d'un fichier de programmes. L'ordinateur les lit sur la disquette avant de les exécuter. Si la disquette ne comporte pas le fichier de commandes appelé, MSX-DOS ne pourra ni le trouver ni l'exécuter, ce qui amènera le message suivant:

```
Bad command or filename
```

Tous les fichiers munis de l'extension .BAT sont des fichiers de commandes, c'est-à-dire des fichiers comportant un certain nombre de commandes MSX-DOS exécutées en séquence.

Tous les fichiers de commandes munis de l'extension .COM sont des fichiers de commandes dans d'autres langages de programmation, tels le langage machine, etc.

Comme les fichiers de commandes avec l'extension .BAT se composent de commandes MSX-DOS, vous pouvez vous-même les créer et les ajouter au système

Les commandes suivantes sont disponibles en MSX-DOS et sont données ici, à titre de référence, avec une courte explication.

| | |
|---|---|
| BASIC | Aller en MSX-BASIC |
| COPY | Copier des fichiers spécifiés |
| DATE | Afficher et changer la date |
| DEL | Effacer des fichiers spécifiés |
| DIR | Afficher les fichiers de la disquette, éventuellement spécifiés |
| ERASE | (voir DEL) |
| FORMAT | Formater une disquette neuve pour MSX |
| MODE | Régler la largeur en caractères à l'écran |
| PAUSE | Introduire une pause avant l'entrée dans un fichier de commandes |
| REM | Donner un message dans un fichier de commandes |
| REN | Changer le nom de fichier |
| RENAME | (voir REN) |
| TIME | Afficher et changer l'heure (optionnelle) |
| TYPE | Afficher le contenu d'un fichier spécifié |
| VERIFY | Mettre en marche et arrêter une vérification |

**Signes de séparation**

Une commande MSX-DOS doit être séparée des nombres ou noms de fichier qui suivent éventuellement. Comme nous l'avons déjà vu en appelant le répertoire, on peut indiquer après cette commande une autre disquette ou unité de disques. Dans ce manuel nous utilisons toujours *l'espace* comme signe de séparation; plusieurs espaces sont également admis.

Ceci pour éviter la confusion avec d'autres signes de séparation, qui sont admis dans certaines commandes et ne le sont pas dans d'autres. Il est donc recommandé d'utiliser seulement l'espace. Pour être complets, nous vous donnons l'ensemble des signes de séparation.

| | |
|---|---|
| '=' | le signe égal à |
| ',' | la virgule |
| ';' | le point-virgule |
| ' ' | l'espace (un ou plusieurs) |
| 'T' | le TAB |

Ces signes de séparation ne sont pas utilisables dans toutes les commandes. Ils sont en principe admis entre les spécifications de différents noms de fichier, d'appareils, etc. Nous vous conseillons cependant de toujours utiliser des espaces pour séparer les commandes et les noms de fichier et pour avoir un bon aperçu de la commande donnée.

**Commandes MSX-DOS**

**BASIC**

Cette commande sert à sortir de MSX-DOS et pour passer en MSX-Disk BASIC. Si l'on tape après cette commande un nom de fichier d'un programme BASIC, ce programme BASIC sera chargé et exécuté automatiquement lorsque BASIC aura été activé.

Exemple: `BASIC TESTFILE.BAS`

Lorsque la ROM BASIC sera active, le programme TESTFILE.BAS sera chargé et exécuté automatiquement. Il faut spécifier le nom de fichier complet, avec l'extension. L'emploi de caractères de substitution n'est pas permis dans cette commande.

N'oubliez pas que la division de la mémoire de l'ordinateur change avec cette commande, puisque la division pour MSX-Disk BASIC est différente de celle pour MSX-DOS.

Pour revenir en MSX-DOS, il faut donner la commande 'CALL SYSTEM' ou sa version abrégée '–SYSTEM'.

---

**EN RESUME**

1) BASIC
2) BASIC <nom de fichier>

1) Sortir de MSX-DOS pour passer en MSX-Disk BASIC. Pour revenir en MSX-DOS, utiliser la commande BASIC 'CALL SYSTEM'. Ne pas oublier que la disquette par défaut doit être munie dans ce cas des deux fichiers DOS pour pouvoir charger à nouveau MSX-DOS. L'appel du système (SYSTEM CALL) fait partie de MSX-DOS, ce qui signifie que cet appel ne fonctionne que si l'ordinateur a été démarré avec MSX-DOS.
2) Sortir de MSX-DOS, passer en MSX-Disk BASIC et exécuter automatiquement le fichier de programme BASIC spécifié.

---

**COPY** Cette commande sert à copier un ou plusieurs fichiers d'une disquette sur une autre. Cette commande fonctionne aussi bien avec une seule unité de disquettes qu'avec deux.

Exemple: `COPY TESTFILE.BAS TEST.BAS`

Cette commande copiera le fichier TESTFILE.BAS sur la même disquette sous un autre nom, à savoir TEST.BAS. Copier un fichier sur la même disquette sous le même nom est impossible. La duplication terminée, le signal suivant apparaît:

```
1 files copied
```

Le deuxième nom est une option et peut être remplacé par une lettre pour une autre disquette ou une autre unité de disquettes. Le fichier est alors copié sous un autre nom sur une autre disquette.

```
COPY TESTFILE.BAS B:
```

copiera le fichier TESTFILE.BAS de la disquette par défaut sur la disquette B sous le même nom.

Pour copier d'une disquette sur la disquette par défaut on utilise la commande:

```
COPY C:TESTFILE.BAS
```

Le fichier TESTFILE.BAS est maintenant copié de la disquette C sur la disquette par défaut. Si l'on veut copier d'une disquette sur une autre disquette, il faut les spécifier toutes les deux:

```
COPY C:TESTFILE.BAS D:
```

ou bien

```
COPY C:TESTFILE.BAS D:TEST.BAS
```

Dans le premier cas le fichier est copié et enregistré sous le même nom sur la disquette D, et dans le deuxième cas le fichier aura sur la disquette D le nom TEST.BAS.

Lors de la duplication il est également possible de combiner plusieurs fichiers en un seul. Les fichiers à copier sont spécifiés, séparés par le signe '+'.

```
COPY A:TEST1.TXT + B:TEST2.TXT A:TOTAL.TXT
```

Les fichiers TEST1.TXT de la disquette A et TEST2.TXT de la disquette B sont combinés en un seul nouveau fichier sur la disquette A sous le nom TOTAL.TXT.

Plusieurs fichiers de textes peuvent ainsi etre combinés en un. Dans l'exemple précédent TEST2.TXT sera enregistré après TEST1.TXT dans le nouveau fichier.

Normalement seuls les fichiers ASCII sont combinés. Les fichiers ASCII sont en général des fichiers de textes créés par un éditeur de textes, ou des programmes BASIC sauvegardés par l'option A.

La fin d'un fichier ASCII se caractérise par un code 'CONTROL-Z' (∧ Z; c'est le code du caractère 26 en MSX).

Les fichiers binaires et fichiers de programmes n'ont pas de code ∧ Z à la fin. Il faut consult er le répertoire pour la longueur d'un tel fichier, à l'aide de l'option '/B'. Exemple:

```
COPY/B TESTA.COM + TESTB.COM
```

indique qu'il faut consulter le répertoire pour la longueur des fichiers TESTA.COM et TESTB.COM. Dans cet exemple le fichier TESTB.COM est mis après TESTA.COM et le nouveau fichier s'appelle TESTA.COM. Le fichier TESTA.COM original n'existe donc plus et a été remplacé par une combinaison des deux fichiers.
On utilise l'option '/A' pour ajouter un code ∧ Z à un fichier. Exemple:

```
COPY TESTA.COM/B TESTB.COM/A
```

indique qu'il faut consulter le répertoire pour la longueur du fichier TESTA.COM. La duplication terminée, un code ∧ Z est ajouté au nouveau fichier TESTB.COM.

L'option '/A' ajoutera un seul code ∧ Z au nouveau fichier. Si un deuxième code ∧ Z est à ajouter au fichier, l'exemple suivant peut être utile:

```
COPY TEST1.ASC/B TEST2.ASC/A
```

Dans cet exemple le répertoire est consulté pour la longueur du fichier ASCII TEST1.ASC, c'est-à-dire y compris le code ∧ Z normal qui sert de signe de fin de fichier du fichier TEST1.ASC. Aprés la duplication un code ∧ Z est ajouté au nouveau fichier TEST2.ASC. Ce nouveau fichier a maintenant deux codes ∧ Z à sa fin: le premier du fichier TEST1.ASC original et le deuxième grâce à l'option /A.

L'option (/B ou /A) reste active jusqu'à ce que le système rencontre une nouvelle option. Exemple:

```
COPY TESTA.COM/B + TESTB.COM TESTC.COM/A
```

Dans cet exemple le répertoire est consulté pour la longueur des fichiers TESTA.COM et TESTB.COM. Un code ∧ Z est ajouté au nouveau fichier TESTC.COM.

L'utilisation des caractères de substitution est autorisée pour copier des fichiers. Quelques exemples:

```
A) COPY *.TXT COMBI.TEX
B) *.TXT + *.TEX COMBI.TTT
C) COPY *.TXT TOUT.TXT
D) COPY TOUT.TXT + *.TXT
```

A) Tous les fichiers de la disquette par défaut avec l'extension .TXT seront transférés vers un nouveau fichier COMBI.TEX sur cette même disquette.
B) Tous les fichiers de la disquette par défaut avec l'extension .TXT seront combinés à tous les fichiers avec l'extension TEX et mis dans un nouveau fichier COMBI.TTT sur cette même disquette.
C) Cette commande donne un message d'erreur si le fichier TOUT.TXT existe déjà.
D) Par cette commande tous les fichiers de textes avec l'extension .TXT seront combinés et copiés sur le fichier TOUT.TXT déjà existant.

---

**EN RESUME**

La commande COPY consiste en la commande générale suivante:

COPY <unité><nom de fichier> <fusionner>
1 2 3

<unité><nom de fichier>
4 5

1) Si la disquette ou l'unité n'est pas spécifiée, le fichier sera copié à partir de la disquette par défaut.
2) Le nom de fichier doit être spécifié en entier, donc avec l'extension. L'utilisation de caractères de substitution est autorisée. *.* copiera donc tous les fichiers.
3) La fusion de plusieurs fichiers se fait au moyen du signe '+'. S'il n'y a pas de combinaison, les noms sont séparés par un espace.
4) La copie du fichier est faite sur la disquette ou l'unité spécifiée. Si on oublie de spécifier, la copie sera enregistrée sur la disquette par défaut. Si le fichier à copier et la copie sont sur la même disquette, le nom de fichier doit être différent. Il est impossible de copier un fichier sur lui-même.
5) Nom de fichier de la copie. Celui-ci peut être omis si la copie doit avoir le même nom que l'original, mais la copie doit alors être faite sur une autre disquette.

---

**COPY** *Cette commande peut aussi être utilisée pour copier des fichiers sur d'autres appareils*

Exemple: `COPY TEXTE.TXT CON`

Cette commande copiera le contenu du fichier TEXTE.TXT sur CON (la CONsole = l'écran). Le texte contenu dans ce fichier sera affiché. Au lieu de CON vous pouvez spécifier nimporte quel autre appareil:

```
COPY TEXTE.TXT LST
```

fera sortir le fichier de textes sur l'imprimante. De même avec PRN.

```
COPY CON TEST.TES
```

copiera tout ce que vous tapez au clavier sur la disquette par défaut sous le nom TEST.TES et ceci seulement après la fermeture par un caractère ∧ Z. Vous pourrez lire davantage au sujet du transfert de fichiers dans le passage consacré aux fichiers de commandes.

---

**EN RESUME**

COPY <con> <unité><nom de fichier>
1 2 3
COPY <unité><nom de fichier> <con>
4 5 6

1) AUX renvoie à un appareil d'entrée branché et CON au clavier comme organe d'entrée.
2) Lorsque le nom d'une disquette ou d'une unité est spécifié, tout sera copié sur la disquette spécifiée. Si le nom n'est pas spécifié, tout sera copié sur la disquette par défaut.
3) Il faut toujours indiquer un nom de fichier, sinon le système copie sur lui-même et le texte sera affiché une fois de plus.
4) Si la disquette n'est pas spécifiée, le système supposera qu'il s'agit de la disquette par défaut.
5) Les règles de duplication des fichiers restent les mêmes si on utilise des caractères de substitution.

```
COPY * .TXT CON
```

affiche donc successivement tous les fichiers de textes.
6) Spécifie sur quel appareil branché le fichier doit être copié. Ceci peut être un appareil quelconque (AUX), l'imprimante (LST ou PRN) ou, évidemment, l'écran (CON).

---

**DATE** Cette commande sert à contrôler et à changer la date. Le système enregistrera la dernière date entrée dans le répertoire pour chaque fichier sur la disquette, et ceci aussi bien pour de nouveaux fichiers que pour des fichiers corrigés.

Exemple: `DATE <mm> - <jj> - <aa>`

L'ordre peut être différent dans certains pays.

Après avoir donné la commande DATE, vous pouvez directement entrer la date sans utiliser la touche <RETURN>. Si vous appuyez d'abord sur la touche RETURN, le message suivant apparaîtra sur l'écran:

```
Current date is <jour> <m>-<j>-<a>
Enter new date: ■
```

A la première ligne vous voyez la date déjà entrée au préalable. Si vous n'avez pas entré de date lors du démarrage de MSX-DOS, vous voyez:

```
Current date is Sun 1-01-1984
Enter new date: ■
```

Si vous ne voulez pas changer la date, appuyez sur <RETURN> et le curseur par défaut réapparaîtra. Vous pouvez entrer la date immédiatement après la commande DATE. De cette façon l'ordinateur ne demandera pas de nouvelle date.

La nouvelle date doit être entrée en chiffres. Vous avez les possibilités suivantes:

| | | |
|---|---|---|
| <mm> | mois | 1 - 12 ou 01 - 12 |
| <jj> | jour | 1 - 31 ou 01 - 31 |

| | | |
|---|---|---|
| <aa> | année | 0 - 79 ou 00 - 79 |
| | ou | 80 - 90 |
| | ou | 1980 - 2099 |

Les chiffres entrés doivent être séparés par un des signes de séparation suivants:

'-' trait d'union
'.' point
'/' barre oblique

| | |
|---|---|
| 11-28-1990 | devient Jeudi 28 novembre 1990 |
| 4/18/84 | devient Jeudi 18 avril 1984 |
| 06.20.72 | devient Lundi 20 juin 2072 |

L'entrée d'une date erronée amène le message:

```
Invalid date
Enter new date: ■
```

---

**EN RESUME**

1) DATE
2) DATE <mm>-<jj>-<aa>

1) Pour vérifier et changer la date.
2) Pour changer la date

---

**DEL** Cette commande sert à effacer (en anglais: DELete) un ou plusieurs fichiers de la disquette. Le fichier à effacer est spécifié après la commande.

Exemple: `DEL TESTFILE.TXT`

Le fichier TESTFILE.TXT est effacé de la disquette. Si l'unité n'est pas spécifiée, ce sera celle de la disquette par défaut. Si la disquette ou l'unité est spécifiée devant le nom de fichier, le fichier de la disquette concernée sera effacé. Par exemple:

```
DEL B:TESTFILE.TXT
```

L'utilisation de caractères de substitution est permise, bien qu'il faille être très prudent avec eux.

```
DEL B:*.TXT
```

Cette commande effacera tous les fichiers avec l'extension .TXT sur la disquette B. Pour effacer tous les fichiers de la disquette, on peut utiliser la commande suivante:

```
DEL B:*.*
```

Pour éviter les erreurs l'ordinateur demande si cette commande d'effacement total doit bien être exécutée.

```
Are you sure (Y/N) ■?
```

(êtes-vous sûr (oui/non))

Taper 'n' ou 'N' annulera la commande, taper 'y' ou 'Y' ou appuyer sur <RETURN> effacera tous les fichiers sur la disquette B.
Au lieu de la commande DEL on peut utiliser ERASE.

---

**EN RESUME**

DEL <unité><nom de fichier>

Si l'unité n'est pas spécifiée, le fichier spécifié sera effacé sur la disquette par défaut. Le nom de fichier doit toujours être spécifié avec l'extension. L'utilisation de caractères de substitution est permise.

---

**DIR** Cette commande fait apparaître le repertoire ( en anglais: DIRectory) de la disquette. Vous pouvez l'obtenir intégralement, les données des fichiers comprises, ou en avoir un abrégé avec seulement les noms des fichiers.

Exemple: `DIR`

L'écran affiche tous les fichiers avec le nombre d'octets qu'ils occupent, la date de leur dernière écriture sur disquette et, eventuellement l'heure (ceci n'est pas possible sur tous les ordinateurs MSX).

```
DIR TESTFILE.TXT
```

Affiche toutes les données du fichier TESTFILE.TXT. Cette commande offre les possibilités suivantes:

| | |
|---|---|
| DIR | identique à DIR *.* |
| DIR <nom de fichier> | identique à DIR <nom de fichier>.* |
| DIR .<extension> | identique à DIR *.<extension> |
| DIR . | identique à *. |

Si vous voulez consulter le répertoire d'une autre disquette que la disquette par défaut, vous devez spécifier cette disquette ou cette unité après la commande:

```
DIR B:
```

Celle-ci fait donc apparaître le répertoire de la disquette B. Après l'affichage de ce répertoire le curseur par défaut réapparaîtra.

Vous pouvez de plus appeler une version abrégée du répertoire au moyen de la commande suivante:

```
DIR /W
```

Ce qui affiche seulement le nom de fichier avec l'extension en deux colonnes, en principe avec autant de noms possibles sur une ligne. Si vous avez plus de 46 fichiers sur une disquette, le premier fichier disparaîtra de l'écran avant que le dernier soit affiché. Pour éviter cela, la commande possède le mode page:

```
DIR /P
```

affiche un maximum de 23 fichiers. S'il se trouve un plus grand nombre de fichiers sur la disquette, le message suivant est donné:

```
Strike a key when ready . . . ■
```

Appuyez sur n'importe quelle touche et le système lira le reste du répertoire et l'affichera.

---

**EN RESUME**

DIR <unité><nom de fichier><mode>
1 2 3

1) Si vous voulez voir les données d'un ou de plusieurs fichiers d'une autre disquette que la disquette par défaut, vous devez indiquer cette disquette.
2) Le nom de fichier est une option qui n'est seulement utile que si vous voulez voir toutes les données d'un certain fichier. Si le nom de fichier est omis, le répertoire entier de la disquette par défaut ou de la disquette spécifiée sera affiché. L'utilisation de caractères de substitution dans le nom de fichier est permise.
3) Si vous tapez '/W' les noms de fichiers seront affichés les uns à côté des autres sur l'écran. Si vous tapez '/P', le nombre de fichiers affichés sera limité à 23.

---

**FORMAT** Cette commande prépare la disquette à la réception et à la mémorisation des fichiers MSX-DOS.

Exemple: `FORMAT`

Suivez les instructions à l'écran et attendez jusqu'à ce que le message suivant apparaisse:

```
Format complete
```

Vous pouvez utiliser en MSX-DOS les disquettes que vous avez formatées sous MSX-Disk BASIC. Le formatage est déterminé par l'unité de disques et le format sera donc le même en MSX-DOS et en MSX-Disk BASIC.

---

**EN RESUME**

FORMAT

Prépare les disquettes pour les fichiers MSX-DOS. Après la commande vous avez la possibilité d'insérer une disquette et de spécifier une unité de disquettes.

---

**MODE** Permet de régler la largeur en caractères de l'écran.

Exemple: `MODE 40`

Par cette commande vous réglez l'écran sur une largeur de 40 caractères. Vous pouvez indiquer chaque largeur entre 1 et 80. En fonction de la largeur en caractères, l'écran est mis en mode de textes ou en mode graphique. Ce mode d'écran est déterminé par le nombre de caractères par ligne Le mode graphique est obtenu par une largeur de 32 caractères ou moins. Une largeur de 33 à 80 caractères mettra l'écran en mode de textes.

Le mode graphique vous permet d'afficher entièrement et correctement sur l'écran l'ensemble des caractères du clavier.

Le mode de textes convient le mieux, évidemment, pour l'affichage de textes. Votre ordinateur MSX a un réglage standard, à savoir:

```
MODE 37
```

Ce qui implique un mode texte avec une largeur de 37 caractères (ou colonnes).

**EN RESUME**

MODE <largeur>

Après cette commande vous pouvez spécifier le nombre de caractères affiché par ligne l'écran. Le réglage standard est le mode de textes d'une largeur de 37 caractères.

---

**REN** Change le nom (en anglais REName) d'un fichier à spécifier par un nouveau nom de fichier spécifié.

Exemple: `REN TEXTE.TXT TEXTE1.TXT`

Le fichier existant TEXTE.TXT sur la disquette par défaut reçoit le nouveau nom TEXTE1.TXT. Le fichier continue à exister mais son nom dans le répertoire est changé. Lorsque le fichier dont le nom doit être changé ne se trouve pas sur la disquette par défaut, il faut spécifier la disquette ou l'unité devant le premier nom de fichier:

```
REN B:TEXTE.TXT TEXTE1.TXT
```

Le fichier sur la disquette B changera maintenant de nom. La spécification d'un nom d'unité devant le deuxième nom de fichier sera ignorée et le fichier sera changé de nom sur la première disquette spécifiée ou sur la disquette par défaut.

Il est permis d'utiliser des caractères de substitution dans le changement de noms de fichier.

```
REN *.TXT *.TST
REN ABCDE B??B?
```

Dans le premier cas tous les noms de fichier avec l'extension .TXT seront remplacés par les mêmes noms de fichier, mais avec maintenant l'extension TST. Par la deuxième commande le fichier ABCDE sera changé en BBCBE; à l'endroit du caractère de substitution (?) les mêmes lettres seront maintenues. A l'endroit où une lettre est spécifiée, l'ancienne lettre sera remplacée par la nouvelle. Les deux commandes s'appliquent à la disquette par défaut.

Lorsqu'on spécifie pour le nouveau fichier un nom de fichier identique à celui d'un fichier déjà présent sur la disquette, un message d'erreur suivra:

```
Rename error
```

Vous recevrez le même message d'erreur lorsque vous voudrez changer le nom d'un fichier qui ne se trouve pas sur la disquette.

**EN RESUME**

REN <unité><ancien nom de fichier> <nouveau nom de fichier>

1 2 3

1) L'unité comportant l'ancien fichier doit être spécifiée, à moins que le fichier se trouve sur la disquette par défaut.
2) Le fichier qui doit changer de nom doit être spécifié avec l'extension. L'utilisation de caractères de substitution est permise: tous les fichiers satisfaisant à la condition changent alors de nom.
3) Le nouveau nom de fichier complet, donc avec l'extension éventuelle, doit être spécifié. L'utilisation de caractères de substitution est permise. L'ancien nom ne changera pas à l'endroit des caractères de substitution.

La commande RENAME (nom complet de REN) est également permise.

---

**TIME** Cette commande sert à vérifier et à changer l'heure. Le système inscrira dans le répertoire l'heure à chaque fichier que vous mettez sur disquette. Ceci est valable pour les nouveaux fichiers et pour les fichiers corrigés.

Exemple: `TIME <hh><:mm><:ss>`

Après la commande TIME vous pouvez entrer immédiatement la nouvelle heure. Si vous ne le faites pas, le système affichera l'heure exacte (si celle-ci a été entrée au préalable) et demandera une heure nouvelle:

```
Current time is: <h>:<m>:<s> <c>
Enter new time: ■
```

Si vous ne voulez pas changer l'heure, appuyez sur <RETURN>. L'heure est entrée seulement en chiffres. Entre les heures, les minutes et, éventuellement, les secondes il faut mettre deux points (":").

| | |
|---|---|
| hh | 00 - 24 |
| mm | 00 - 59 |
| ss | 00 - 59 |
| cc | 00 - 99 |

Les secondes et les centièmes de seconde sont optionnelles et peuvent être omis. Ils ne sont pas séparés par deux points, mais par un seul (".").

MSX-DOS inscrira l'heure d'entrée comme la nouvelle heure si les chiffres et les signes de séparation entrés sont corrects. Sinon, le message suivant apparaîtra sur l'écran:

```
Invalid time           (Heure non valable)
Enter new time: ■      (Entrez nouvelle heure)
```

MSX-DOS attend maintenant que vous tapiez l'heure correcte ou que vous appuyiez sur la touche <RETURN>.
Si votre ordinateur n'est pas muni d'une horloge incorporée, la commande ne fonctionnera pas.

---

**EN RESUME**

TIME <hh><:mm><:ss><.cc>
1 2 3 4

1) Les heures sont spécifiées par un ou deux chiffres.
2) Les minutes, séparées par deux points, doivent être spécifiées quand on a spécifié les heures. Il est impossible de spécifier seulement les heures et ceci amènera à un message d'erreur. Les minutes doivent aussi être spécifiées par un ou deux chiffres.
3) Les secondes peuvent être omises lors de l'entrée d'une nouvelle heure. On suppose alors que ss=00.

---

**TYPE** Affiche le contenu d'un fichier spécifié.

Exemple: `TYPE TEXTE.TXT`

Le contenu du fichier TEXTE.TXT est maintenant affiché. La commande DIR permet de rechercher le nom de fichier correct, si nécessaire.

Des caractères de tabulation éventuels sont convertis en espaces jusqu'à la première colonne de tabulation standard de l'écran. Les taquets de tabulation de l'écran se trouvent toutes les 8 colonnes.

L'affichage d'un fichier binaire sera couplé avec la transmission de tous les caractères de commande éventuels de ce fichier binaire.

**EN RESUME**

TYPE <unité><nom de fichier)
1 2

1) Lorsque vous ne spécifiez pas L'unité de disquettes, le fichier sera lu et affiché à partir de la disquette par défaut.
2) Le nom de fichier doit toujours être spécifié avec l'extension. Il est recommandé de ne pas utiliser les caractères de substitution. La commande prend alors le premier fichier correspondant à la spécification et après son exécution s'arrête. Il n'affichera pas les fichiers suivants qui satisfont également à la spécification avec le caractère de substitution.

---

**VERIFY** Cette commande permet de vérifier toutes les données écrites sur disquette.

Exemple: `VERIFY ON`

Le mode de vérification est mis en marche. Toutes les données enregistrées sur la disquette sont relues et contrôlées par rapport à celles de l'ordinateur. S'il s'est produit une erreur dans l'écriture sur disquette, un message d'erreur apparaît sur l'écran:

```
DISK I/O error
```

Le mode de vérification est abandonné par la commande:

```
VERIFY OFF
```

Après le démarrage le mode de vérification est toujours sur (OFF).

---

**EN RESUME**

VERIFY < on / off >

Vous avez, avec la commande VERIFY, le choix entre ON (contrôle) et OFF (non contrôle). La position standard est OFF.

---

**PAUSE** est une commande utilisée dans un fichier de commandes pour introduire une pause, pour changer de disquette ou pour interrompre l'exécution à ce point.

Exemple: `PAUSE Introduire la disquette B`

Après la commande PAUSE vous pouvez entrer un message, qui sera alors affiché lors de l'exécution du fichier de commandes. Comme signe de séparation entre PAUSE et le message, vous pouvez utiliser l'espace, le caractère de tabulation ou la virgule. Après chaque commande PAUSE la ligne suivante apparaît automatiquement sur l'écran:

```
Strike a key when ready . . . ■
```

(Appuyez sur une touche quand vous êtez prêt.)

Le message introduit éventuellement après la commande PAUSE est d'abord affiché, suivi du message d'appuyer sur une touche.

Chaque touche peut être utilisée pour faire avancer l'exécution du fichier de commandes, à l'exception de <CONTROL>-C.

Lorsque vous appuyez sur <CONTROL>-C, la question suivante apparaît:

`Terminate batch file (Y/N)? ■` (Avez vous terminé le fichier)

Si vous tapez (Y), le fichier de commandes ne sera plus exécuté et le système reviendra en MSX-DOS. Vous pouvez donc utiliser la commande PAUSE pour interrompre prématurément un fichier de commandes à certains endroits.

---

**EN RESUME**

PAUSE [message]

Après la commande PAUSE dans un fichier de commandes, le système d'exploitation attend que vous appuyiez sur une touche.

Avec <CONTROL-C> vous pouvez interrompre l'exécution du fichier de commandes.

Le message après la commande est toujours affiché avant celui d'appuyer sur une touche.

---

**REM** La commande REM (REMarque) permet d'inscrire dans un fichier de commandes des commentaires et des messages, qui seront affichés lors de l'exécution de ce fichier de commandes.

Exemple: `REM Ce fichier copie automatiquement MSX-DOS`

Lorsque vous appelez le fichier de commandes, le message après REM est affiché en premier (à condition qu'il ait été entré

à la première ligne dans le fichier de commandes) ce qui permet de savoir ce que le fichier de commandes va exécuter.

---

**EN RESUME**

REM <message>

Après la commande REM vous pouvez inscrire dans un fichier de commandes des messages à afficher.

---



# FICHIER DE COMMANDES

**Introduction**

Un fichier de commandes est un fichier qui contient des commandes MSX-DOS combinées et enregistrées en un seul fichier. Le nom de ce fichier de commandes sans extension est la commande par laquelle l'ordinateur va lire et exécuter le fichier de commandes. En MSX-DOS on utilise l'extension .BAT.

Un tel lot (BATCH) de programmes peut s'intituler par exemple NOUVEAU; sur la disquette ce fichier sera indiqué comme suit:

```
NOUVEAU.BAT
```

Ce fichier de commandes peut par exemple formater une disquette neuve, puis la munir de tous les fichiers MSX-DOS pour le système d'exploitation.

Si vous travaillez surtout en Disk-BASIC, vous pouvez créer un fichier de commandes qui assure que vous entrez en BASIC et qui ensuite affiche automatiquement un menu.

Il est également possible de créer un fichier de commandes qui s'exécutera automatiquement après le démarrage: c'est le fichier de commandes AUTOEXEC. Si la disquette comporte un fichier avec le nom AUTOEXEC.BAT, ce fichier sera toujours exécuté automatiquement après le démarrage.

**Qu'est-ce qu'un fichier de commandes?**

Comme on l'a dit plus haut, un fichier de commandes comporte un certain nombre de commandes MSX-DOS, qui exécutent ensemble une certaine fonction. Une telle fonction intervient chaque fois qu'on doit entrer des commandes particulières. MSX-DOS permet d'entrer ces commandes dans un fichier spécial, qui s'appelle le fichier de commandes. Quand vous aurez tapé le nom de fichier (sans extension), le système d'exploitation lira le fichier de la disquette et exécutera les commandes particulières, comme si vous les spécifiiez vous-même à l'ordinateur au moyen du clavier. Chaque fichier de commandes sur la disquette doit avoir l'extension .BAT. Il est exécuté quand le nom de fichier est tapé sans cette extension.

Vous pouvez constituer un fichier de commandes à l'aide du programme EDITOR ou à l'aide de la commande COPY. Il y a deux commandes MSX-DOS spéciales pour l'usage avec un fichier de commandes: REM et PAUSE. REM permet d'afficher des messages et des instructions lors de l'exécution du fichier de commandes. La commande PAUSE a deux fonctions: la première vous permet d'interrompre le fichier de commandes à chaque commande PAUSE et la seconde vous permet d'utiliser la commande PAUSE pour changer de disquette.

Un fichier de commandes peut être très utile: regardez seule-

ment toutes les commandes qui sont nécessaires pour préparer une disquette neuve pour MSX-DOS et la munir des deux fichiers MSX-DOS.

Si nous énumérons les commandes nécessaires, nous obtenons le résultat suivant:

```
FORMAT

COPY A:COMMAND.COM B:

COPY A:MSXDOS.SYS B:

DIR B:
```

Après l'entrée et l'exécution de ces commandes, la disquette est prête à l'emploi. Evidemment, il est beaucoup plus facile de combiner ces commandes dans un fichier de commandes qui les exécutera toutes pour nous.
Le contenu d'un tel fichier de commandes peut être le suivant:

1: REM Ce fichier formate une disquette neuve
2: REM et la munit des fichiers MSX-DOS
3: PAUSE Insérez la disquette neuve dans B:
4: FORMAT
5: PAUSE Transférer MSX-DOS?
6: COPY A: COMMAND.COM B:
7: COPY A: MSXDOS.SYS B
8: DIR B:

Si nous mettons ce fichier de commandes sur notre disquette système de sécurité sous le nom NOUVEAU.BAT, nous pouvons, pour préparer une disquette neuve, l'insérer dans l'unité B et insérer la disquette système dans l'unité A. Lorsque nous n'avons qu'une seule unité, nous commençons par mettre la disquette système dans l'unité A. Lors de l'exécution du fichier de commandes le système nous indiquera quand la disquette neuve (B) est à insérer et à retirer. Après l'entrée de la commande 'NOUVEAU', les trois premières lignes seront affichées ainsi que l'instruction:

```
Strike a key when ready . . .
```

Toutes les commandes seront ainsi executées, jusqu'à l'affichage du répertoire de la disquette B.

L'exécution des commandes du fichier terminée, le curseur par défaut reviendra pour attendre une nouvelle commande.

Ci-dessous, un certain nombre d'indications importantes pour l'utilisation des fichiers de commandes. Lisez-les attentivement avant de créer vous-même un tel fichier.

1 N'utilisez pas BATCH comme nom de fichier (à moins que le nom du fichier que vous désirez exécuter soit BATCH.BAT.
2 Tapez seulement le nom du fichier de commandes à insérer, sans l'extension .BAT.
3 Les commandes dans le <fichier de commandes> .BAT seront alors exécutées.
4 Lorsque vous appuierez sur <CONTROL>-C lors de l'exécution d'un programme séquentiel, le système vous posera la question:

```
Terminate batch file (Y/N)? ■
```

Si vous tapez 'Y', le système arrêtera l'exécution du fichier de commandes et le curseur par défaut réapparaîtra.
Si vous tapez 'N', le système continuera avec la commande suivante. La commande qui était exécutée quand vous avez tapé <CONTROL>-C est interrompue et n'est plus exécutée. Utilisez la commande PAUSE pour interrompre l'exécution d'un fichier de commandes et éventuellement en sortir.
5 Si vous enlevez la disquette comportant le fichier de commandes de l'unité lors de son exécution, le système vous demandera de réinsérer la disquette:

```
Insert disk with batch file
and strike any key when ready
```

S'il n'y a pas de disquette dans l'unité, le système affichera le message suivant:

```
Not ready error reading drive A
Abort, Retry, Ignore?
```

Après l'insertion de la bonne disquette, la commande suivante sera lue et exécutée.
6 La dernière commande dans un fichier de commandes peut être le nom d'un autre fichier de commandes. Vous appelez ainsi un fichier de commandes suivant dès que le fichier de commandes précédent est terminé.

**Fichiers AUTOEXEC.BAT**

Une forme particulière du fichier de commandes est celle du fichier AUTOEXEC.BAT qui, comme l'indique son nom, est exécuté automatiquement. Quand MSX-DOS est démarré, le système regarde s'il y a sur la disquette un fichier avec le nom AUTOEXEC.BAT.

Si un tel fichier s'y trouve, l'entrée de la date sera ignorée et le fichier AUTOEXEC.BAT est exécuté. Cette exécution automatique d'un fichier de commandes est trés facile si vous avez une disquette avec un seul programme important que vous voulez faire exécuter (en anglais: RUN) automatiquement dès que vous démarrez l'ordinateur avec cette disquette.

Lorsque le système ne trouve pas de fichier AUTOEXEC.BAT, il demande la date et l'heure.

Naturellement un fichier AUTOEXEC.BAT peut aussi renvoyer à une autre disquette ou une autre unité. L'exécution automatique d'un programme BASIC après le démarrage est également possible.

Votre disquette système comporte un fichier AUTOEXEC.BAT, qui donnera l'aide à l'utilisateur immédiatement après le démarrage. Cette aide à l'utilisateur vous permet d'entrer la date correcte par le choix 10.

**Créer un fichier de commandes**

Pour faire exécuter automatiquement un programme BASIC après le démarrage, on crée un fichier AUTOEXEC.BAT. Ceci est simple, bien sûr, avec la commande COPY, mais si vous avez un programme EDITOR, il est plus facile d'utiliser celui-ci, puisque vous avez alors des possibilités de correction. Nous utiliserons ici la commande copy, parce que c'est une commande interne standard et toujours présente.

Tout d'abord nous devons indiquer au système que nous allons créer un fichier portant le nom AUTOEXEC.BAT. Le système doit en même temps enregistrer dans ce fichier et mettre sur disquette toutes les commandes que nous donnons. En effet, le fichier de commandes ne peut être exécuté qu'à partir de la disquette; le système lit les commandes une à une et les exécute de la même façon. Nous donnons au système la commande suivante de tout copier dans le fichier à partir du clavier:

```
COPY CON AUTOEXEC.BAT
```

Cette commande dit à MSX-DOS d'enregistrer toute l'information à partir du clavier (entrée) dans le fichier AUTOEXEC.BAT et de la mettre sur disquette (sortie) dès que toutes les commandes ont été données.
Nous allons maintenant entrer la première commande:

```
BASIC
```

Comme vous le savez déjà, cette commande assure que le système sort de MSX-DOS et entre dans MSX-Disk BASIC. Après la commande BASIC vient le nom du programme que vous désirez faire exécuter (RUN) automatiquement:

```
BASIC MENU.BAS
```

Il faut, évidemment, que le fichier MENU.BAS soit présent sur la disquette par défaut.

Comme le fichier AUTOEXEC.BAT est exécuté sous MSX-DOS, on peut également spécifier une autre disquette pour le nom de fichier:

```
BASIC B:MENU.BAS
```

Après cela il ne faut plus de commandes et nous fermons la ligne de commande avec <RETURN>. Pour mettre la ligne de commandes sur disquette, il faut appuyer sur <CONTROL>-Z, puis sur <RETURN>. Le fichier AUTOEXEC.BAT est maintenant enregistré sur disquette. A chaque démarrage de MSX-DOS (avec la disquette comportant ce fichier de commandes), le fichier AUTOEXEC.BAT sera exécuté. L'ordinateur entrera alors en BASIC et le programme MENU sera exécuté automatiquement.

Une remarque importante ici. Si on passe en BASIC par l'intermédiaire d'un fichier AUTOEXEC.BAT, la commande à exécuter doit suivre immédiatement la commande BASIC, seulement séparée de celle-ci par des espaces. Les autres signes de séparation amèneront au message d'erreur:

```
Bad filename
```

Le fichier NOUVEAU.BAT mentionné plus haut peut également être mis sur la disquette système. On peut alors formater rapidement et simplement des disquettes neuves et les munir de fichiers MSX-DOS.

Pour cela nous entrons d'abord la commande COPY, suivie des commandes particulières telles que nous les avons étudiées au paragraphe 7.1:

```
COPY CON NOUVEAU.BAT
REM Ce fichier formate une disquette neuve
REM et la munit des fichiers MSX-DOS
PAUSE Insérez la disquette neuve dans B:
FORMAT
PAUSE Transférer MSX-DOS?
COPY A:COMMAND.COM B:
COPY A:MSXDOS.SYS B:
COPY A:AUTOEXEC .BAT B:
COPY A:DOSHLP.COM B:
DIR B:
^ Z
```

**Fichier de commandes avec variables**

Il se peut que vous ayez une application qui doive souvent se répéter dans d'autres fichiers, par exemple l'addition d'un fichier de textes à un fichier de textes déjà existant.

MSX-DOS a, pour l'utilisation de variables dans un fichier de commandes, 10 paramètres différents: de %0 à %9. Ceux-ci peuvent être remplacés, si c'est nécessaire, par des noms de fichier ou par des nombres.

La création du fichier de commandes est alors, le suivant:

```
A>COPY CON TEXTE.BAT
COPY %1.TXT LST
COPY TOUS.TXT + %1.TXT
DEL %1.TXT
^ Z
```

Le fichier de commandes TEXTE.BAT comporte une variable à remplacer par un nom de fichier. L'exécution du fichier de commandes se fait par la commande:

```
TEXTE CONTENU
```

Le premier mot est évidemment le nom du fichier de commandes, qui est à lire sur la disquette; le deuxième mot est le paramètre mis à la place de %1.
Le fichier de commandes est maintenant le suivant:

| | |
|---|---|
| COPY CONTENU.TXT LST | Le fichier de textes CONTENU.TXT est sorti par l'imprimante. |
| COPY TOUS.TXT + CONTENU.TXT | Le fichier de textes CONTENU.TXT est copié et enregistré après le fichier de textes TOUS.TXT. |
| DEL CONTENU.TXT | Le fichier de textes CONTENU.TXT est effacé de la disquette. |

Le fichier de commandes permet d'ajouter, après sa création, un autre fichier de textes au fichier de textes existant TOUS.TEXT:

```
TEXTE NVTEXTE
```

Le fichier sera maintenant sorti par l'imprimante, enregistré après le fichier de textes TOUS.TXT et enfin effacé.

La variable %0 est la référence au nom du fichier de commandes lui-même, donc sans extension. Toutes les variables doivent être entrées dans l'ordre numérique après le nom du fichier de commandes. Comme exemple, le fichier de commandes suivant avec le nom TEST.BAT:

```
TYPE %2.TXT
COPY %1.TXT LST
TYPE %0.BAT
DIR %3:
```

La commande à entrer est:

```
TEST CONTENU TEXTE B
 %0     %1      %2  %3
```

Les commandes sont entrées dans l'ordre numérique, l'ordre dans le fichier de commandes est sans importance. Comme première commande, le fichier de commandes affichera le fichier de textes TEXTE.TXT. Le fichier de textes CONTENU.TXT sera ensuite imprimé et nous voyons le contenu du fichier de commandes TEST.BAT exécuté et le répertoire de la disquette B.

Il n'est pas obligatoire d'utiliser la variable %0 dans un fi-

chier de commandes, mais les autres variables doivent commencer par %1, et suivre. Si on n'entre que la variable %5 dans un fichier de commandes, le système regarde la cinquième variable après le nom spécifié du fichier de commandes. Si on n'a introduit que le nom et la variable requise, le système ne voit pas de cinquième variable et amène le message:

```
File not found
```

Si on affirme que la variable requise vient en cinquième position, le fichier de commandes fera son travail, par exemple:

```
TEST 1 2 3 4 CONTENU.TXT
```

Le système supposera maintenant que les paramètres 1 à 4 sont bien respectivement les variables %1 à %4, bien qu'elles ne soient mentionnées ni utilisées dans le fichier de commandes. La variable %5 est remplacée par CONTENU.TXT et le fichier de commandes peut faire son travail.

# 7 EDITION MSX-DOS

**Mémoire des commandes**

Quand vous appuyez sur la touche <RETURN> toutes les commandes entrées sont d'abord rangées dans une mémoire des commandes. Cette mémoire s'appelle en anglais 'template'. Nous le signalons, car on rencontre souvent ce nom dans différentes publications. Bien évidemment le système exécutera la commande. Comme la commande se trouve dans la mémoire des commandes, même après son exécution, il y a des touches spéciales pour appeler cette commande. Après appel de la mémoire, la commande sera affichée. Si on appuie une fois de plus sur la touche <RETURN>, la commande est exécutée à nouveau.

Lorsqu'on entre une commande et qu'il s'avère lors de l'exécution qu'elle contient une erreur, des touches spéciales pourvues de fonctions de correction permettent de répéter la commande avec un minimum de dactylographie. En faisant bon usage des fonctions, on n'a qu'à retaper les caractères incorrects.

Après la correction d'une ligne de commande et son exécution, celle-ci est rangée dans la mémoire des commandes. Ainsi la dernière commande donnée est rangée dans la mémoire et disponible pour être utilisée ultérieurement.

**Touches de fonction**

Nous regardons d'abord les touches de commande du curseur, qui assurent une partie des fonctions de correction. La meilleure façon de procéder est d'entrer d'abord une ligne de commande et de la ranger dans la commande des mémoires en appuyant sur <RETURN>. Il faut tester la ligne rangée dans la mémoire des commandes au moyen des fonctions des touches de commande du curseur et vérifier si le fonctionnement correspond.

↓ Montre le contenu de la mémoire des commandes.
↑ Efface de l'écran la ligne de commande qui vient d'être affichée.
→ Montre le caractère suivant de la mémoire des commandes.
← Efface le dernier caractère affiché.

Il y a encore un certain nombre de fonctions spéciales qui permettent d'extraire des parties de la ligne de commande de la mémoire des commandes, d'ajouter de nouvelles commandes et de ranger la nouvelle ligne de commande dans la mémoire des commandes sans l'exécuter au préalable. Cette dernière fonction permet de vérifier et éventuellement de corriger à nouveau la commande. Appuyez sur la touche quand vous êtes sûr que la ligne est correcte. La commande est exécuté quand vous appuyez sur <RETURN>.

| | |
|---|---|
| SELECT <caractère> | Cette fonction fait apparaître le contenu de la mémoire des commandes jusqu'au caractère spécifié. Le caractère spécifié n'est pas affiché. |
| DEL | Saute les caractères dans la mémoire des commandes. Appuyer une seule fois et ensuite appuyer sur la touche ↓ fait apparaître la ligne de commande sans le premier caractère. Quand vous appuyez cinq fois sur la touche les cinq premiers caractères sont sautés. |
| CLR <caractère> | Saute les caractères de la ligne de commande dans la mémoire des commandes jusqu'au caractère spécifié. Quand vous appuyez sur la touche ↓, le reste de la ligne de commande est affiché. |
| INS | Mode d'insertion. Permet d'ajouter dans une ligne de commande des caractères ou des commandes entières entre les commandes existantes. |
| HOME | En appuyant sur la touche HOME, la ligne de commande corrigée est rangée dans la mémoire des commandes.<br>Un '@' se place sur l'écran derrière la nouvelle ligne de commande rangée et le curseur vient se placer sous le premier caractère de cette nouvelle ligne, sans le curseur 'A>' normal. |

Les fonctions décrites ici peuvent être appelées non seulement par les touches qui viennent d'être mentionnées, mais encore par des caractères de commande. Pour certaines fonctions on peut même utiliser plusieurs caractères de commande ou plusieurs touches.

Nous entrons maintenant la commande suivante et appuyons sur <RETURN>:

```
DIR A:PROGRAM.COM        <RETURN>
```

L'écran affichera toute l'information concernant le fichier demandé. La ligne de commande est rangée dans la mémoire des commandes. Pour répéter la commande, appuyez seulement sur la touche ↓ et puis sur <RETURN>.

Cette commande répétée est également affichée à l'écran. Dès que vous avez appuyé sur la touche ↓, la commande entière est affichée et quand vous appuyez sur <RETURN>, la commande est transférée au processeur des commandes pour être exécutée.

Quand vous voulez toutes les données du fichier PROGRAM.TXT qui se trouvent aussi sur la disquette A, vous pouvez utiliser les instructions suivantes:

<SELECT>C    `DIR A:PROGRAM.■`

A droite de l'écran vous voyez une partie de la mémoire des commandes affichée directement quand vous entrez la lettre C. Ne tapez maintenant que les lettres TXT et la ligne de commande affichée devient:

```
DIR A:PROGRAM.TXT■
```

La ligne de commande est maintenant prête à être transférée au processeur des commandes, au moyen de la touche <RETURN> par laquelle le contenu de la ligne de commande est rangée en même temps dans la mémoire des commandes.

Supposons que nous voulions voir à l'écran le contenu d'un même programme se trouvant sur la disquette B. Nous procédons de la manière suivante (à gauche les entréss sur le clavier et à droite l'affichage):

| *CLAVIER* | *ECRAN* |
|---|---|
| 1) TYPE | A>TYPE■ |
| 2) <INS> | A>TYPE■ |
| 3) <ESPACE>B | A>TYPE B■ |
| 4) <DEL> | A>TYPE B■ |
| 5) <touche ↓ > | A>TYPE B:PROGRAM.TXT■ |
| 6) <RETURN> ou <HOME> | |

Que s'est-il passé? Tout d'abord, le mot 'TYPE' a été tapé sur le mot 'DIR' dans la mémoire des commandes. Comme il doit y avoir un espace entre la première commande et le nom de fichier mais qu'il a été supprimé par le 'E' de 'TYPE', nous appuyons d'abord sur 'INS' puis tapons un '<ESPACE>' et ensuite la lettre 'B' pour l'autre disquette. Dans la mémoire des commandes le curseur se trouve toujours devant le 'A', nous l'effaçons en appuyant sur '<DEL>'. Le reste est extrait de la mémoire des commandes par la touche du curseur ↓ et vient se placer après le texte sur l'écran. La commande est maintenant telle que nous la souhaitons. Pour être plus clairs, nous vous montrons, par le détail, ce qui s'est passé, à gauche ce que nous voyons sur l'écran et à droite ce qui se passe dans la mémoire des commandes:

```
                             DIR A:PROGRAM.TXT
1)  TYPE■                    TYPE■A:PROGRAM.TXT
2)  TYPE■                    TYPE■A:PROGRAM.TXT
3)  TYPE B■                  TYPE B■A:PROGRAM.TXT
4)  TYPE B■                  TYPE B■:PROGRAM.TXT
5)  TYPE B:PROGRAM.TXT■      TYPE B:PROGRAM.TXT■
```

Tout ce que vous tapez est donc retape sur la ligne de commande dans la mémoire des commandes, jusqu'à ce que vous appuyiez sur <INSERT>. Tout ce que vous tapez maintenant est

inséré entre ce que vous avez retapé dans la mémoire des commandes et ce qui y reste. Par <DEL> (= DELete = effacer), vous effacez le caractère suivant dans la mémoire des commandes et par la touche ↓ vous transférez le reste de la mémoire des commandes vers la ligne de commande sur l'écran.

En appuyant sur <RETURN> vous transférez la ligne de commande vers le processeur des commandes, qui assure son exécution. Quand vous ne voulez pas que la commande soit exécutée ou quand vous pensez qu'elle contient une erreur, vous appuyez sur la touche <HOME> pour ranger provisoirement la ligne de commande dans la mémoire des commandes. Vous pouvez alors examiner la commande et si vous n'y découvrez pas d'erreur, vous pouvez la faire exécuter en appuyant d'abord sur la touche ↓, ce qui fait réapparaître la ligne de commande sur l'écran, puis sur la touche <RETURN>.
Une deuxième méthode pour modifier cette commande est

| *CLAVIER* | *ECRAN* |
|---|---|
| 1) <INS> | A>■ |
| 2) TYPE B | A>TYPE B■ |
| 3) <CLR>: | A>TYPE B■ |
| 4) <touche ↓> | A>TYPE B:PROGRAM.TXT■ |
| 5) <HOME> ou <RETURN> | |

Vous voyez que cette méthode demande moins d'actions, mais ce qui compte est ce qui est le plus facile en pratique. La seule manière pour bien apprendre à manipuler est de s'exercer aux différentes lignes de commande.

Supposons que nous n'ayons pas correctement exécuté la commande précédente et que la mémoire des commandes affiche la ligne suivante:

```
BYTE B:PROGRAM.TXT
```

Pour corriger, nous pouvons procédér de la manière suivante:

| *CLAVIER* | *ECRAN* |
|---|---|
| 1) <DEL> | A>■ |
| 2) <DEL> | A>■ |
| 3) <touche →> | A>T■ |
| 4) <INS> | A>T■ |
| 5) YP | A>TYP■ |
| 6) <touche ↓> | A>TYPE B:PROGRAM.TXT■ |

Ce n'est qu'un exemple pour montrer quelles fonctions les différentes touches exécutent. Vous voyez clairement qu'appuyer sur la touche <DEL> n'a pas d'effet sur l'écran. L'effet ne peut être constaté que dans la mémoire des commandes que nous ne pouvons pas observer. Ceci vaut bien sûr également pour les

touches <INS>, <SELECT> et <CLR>. En pratique une telle erreur se corrigera plus rapidement et plus facilement de la manière suivante:

| *CLAVIER* | *ECRAN* |
|---|---|
| 1) <INS> | A>■ |
| 2) TYPE | A>TYPE■ |
| 3) <CLR> | A>TYPE■ |
| 3) B | A>TYPE■ |
| 4) <touche ↓ > | A>TYPE B PROGRAM.TXT■ |
| 5) <HOME> ou <RETURN> | |

Vous voyez qu'il y a plusieurs possibilités pour corriger la ligne de commande dans la mémoire des commandes. La méthode la plus facile est de travailler avec les touches <CLR> et <SELECT>, qui pour la première saute les caractères dans la mémoire des commandes et laisse tout derrière le curseur imaginaire être transféré à l'écran par la touche ↓.

Avec <SELECT> vous transférez à l'écran tous les caractères jusqu'à celui où vous avez placé le curseur imaginaire dans la mémoire des commandes.

Le tableau suivant redonne toutes les fonctions de correction, y compris les caractères de commande permettant d'exercer les mêmes fonctions.

| | |
|---|---|
| NOTE: | sur les claviers version France |
| CLR | correspond à SUP. |
| HOME | correspond à DEP. |
| CLS | correspond à EFE. |

Fonctions visibles à l'écran:

| *Touche* | *Description* |
|---|---|
| ↓<br>∧ – | Montre le contenu intégral de la mémoire des commandes |
| ↑<br>ESC<br>∧ U<br>∧ [ | Efface de l'écran le contenu intégral affiché jusqu'ici et ne modifie pas le contenu de la mémoire des commandes |
| →<br>∧ \ | Montre le caractère suivant de la mémoire des commandes |

| | |
|---|---|
| ←<br>∧ H<br>∧ ] | Efface le dernier caractère affiché, tenue appuyée cette touche efface caractère par caractère la ligne de commande entière. |
| HOME<br>∧ K | Transfère la ligne de commande de l'écran à la mémoire des commandes. |

Fonctions invisibles:

| *Touche* | *Description* |
|---|---|
| SELECT<br>∧ X | Transfère tout de la mémoire des commandes à l'écran, jusqu'au caractère spécifié. |
| CLS<br>∧ L | Saute dans la mémoire des commandes les caractères jusqu'au caractère spécifié. |
| DEL | Saute un seul caractère dans la mémoire des commandes. |
| INS<br>∧ R | Enclenche le mode d'insertion, la mémoire des commandes fait de la place pour le nouveau texte. |

*NOTE:*
*Le signe '∧' indique qu'on doit appuyer sur <CTRL> en même temps que le caractère spécifié. '∧ ∧' est identique à <CTRL>∧.*

**Fonctions des caractères de commande**

Outre les caractères de commande mentionnés ci-dessus, cinq autres méritent votre attention particulière. Le plus important d'entre eux est <CONTROL>-C qui interrompt le processeur des commandes et arrête la commande en exécution. Nous avons déjà rencontré cette commande dans le fichier de commandes. Elle interrompt le fichier de commandes après un certain nombre de commandes déjà exécutées, quand on ne désire plus en exécuter le reste.

Dans ce manuel, partout où il est question d'appuyer sur une touche quelconque, il est interdit d'appuyer sur la touche <CONTROL>-C.

<CONTROL>-S arrête l'affichage. Si nous regardons un long fichier de textes avec la commande TYPE, la première ligne du texte aura vite disparu de l'écran. En appuyant sur <CONTROL>-S, vous arrêtez l'affichage jusqu'à ce vous appuyiez sur une touche quelconque. Quand vous appuyez sur <CONTROL>-C, la commande est interrompue et le curseur par défaut réapparaît.

Il est facile d'utiliser <CONTROL >-S comme interrupteur marche/arrêt lors de l'affichage de longs textes, etc. Le premier <CONTROL>-S arrêtera l'affichage et le deuxième le poursuivra.

<CONTROL>-J envoie le curseur au début de la ligne suivante sur l'écran. Ceci est utile surtout dans le cas de ligne de commande longues pour avoir une meilleure compréhension de certaines parties de la commande., <CONTROL>-J affiche seulement et ne change pas le contenu de la mémoire des commandes. L'utilisation de cette fonction vous amènera à oublier les signes de séparation. Si vous utilisez souvent cette fonction, nous vous recommandons de vous habituer à entrer d'abord l'espace bien que ce ne soit pas nécessaire avant d'utiliser <CONTROL>-J.

<CONTROL>-P met en marche l'imprimant branchée et, en écho, mettra sur papier toute l'information de l'écran. Cette fonction est très utile pour imprimer le répertoire (DIR). Il s'avèrera en pratique que l'utilisation de <CONTROL>-P avant la commande DIR est la plus efficace, puisque la commande elle-même est imprimée. Avec <CONTROL>-P après la commande DIR, seul le répertoire sera mis sur papier et vous n'aurez pas de référence quant à la disquette d'où vient le répertoire.

| | |
|---|---|
| `COPY TEXTE.TXT LST` | est identique à: |
| `<CONTROL>-P TYPE TEXTE.TXT` | |
| `<CONTROL>-P DIR:B` | met le répertoire sur papier, |
| `COPY DIR:B LST` | donne le message d'erreur 'File not found' |

<CONTROL>-N arrête l'imprimante et l'écran assure seul l'affichage. Cette fonction ne fonctionne qu'entre les commandes.

L'imprimante ne peut donc être arrêtée qu'après une certaine commande et non pas lors de l'exécution d'un programme ou d'une commande. N'oubliez donc pas d'arrêter l'imprimante à temps.

| *Caractère* | *Description de la fonction* |
|---|---|
| ∧ P | Met en marche l'imprimante |
| ∧ N | Arrête l'imprimante |
| ∧ J | Place le curseur au début de la ligne suivante de l'écran |
| ∧ S | Interrompt temporairement l'affichage |
| ∧ C | Interrompt les commandes et l'exécution des fichiers de commandes. |

# 8 APERÇUS

**Erreurs, codes et signalisations**

MSX-DOS non seulement affiche différents messages, mais enregistre et signale aussi les erreurs lors de l'exécution. Avant d'afficher ce message, MSX-DOS essayera trois fois d'exécuter la commande. S'il y a toujours erreur, MSX-DOS donnera le message suivant:

```
<1> error <2> drive <3>
Abort, Retry, Ignore?■
```

<1> Le message indique ici quel est le problème rencontré sur la disquette ou l'unité. Vous pouvez vous attendre aux trois possibilités suivantes:

| | |
|---|---|
| Write protect | (Il est impossible d'écrire sur la disquette) |
| Not ready | (Il n'y a pas de disquette dans l'unité) |
| Disk | (Se produit quand il est impossible de lire ou d'écrire sur la disquette, par exemple parce que celle-ci n'a pas encore été formatée. |

<2> Ce message indique à quel moment de l'exécution en était la commande quand l'erreur s'est produite. Il n'y a que les deux possibilités suivantes:

| | |
|---|---|
| reading | (lors de la lecture de la disquette) |
| writing | (lors de l'écriture sur la disquette) |

<3> Enfin, MSX-DOS signale sur quelle disquette ou dans quelle unité l'erreur s'est produite (A, B, C ou D).

Sous le message d'erreur MSX-DOS demande ce qu'il doit faire de l'erreur:

A Abort (arrêter). Le programme est interrompu et le curseur par défaut réapparaît sur l'écran.
R Retry (essayer à nouveau). MSX-DOS essayera à nouveau d'exécuter la commande quand vous aurez corrigé l'erreur.
I Ignore (ignorer). MSX-DOS ignorera le secteur de la disquette où l'erreur s'est produite et poursuivra. Vous risquez ainsi de perdre de l'information.

En général on essayera d'abord 'R' pour voir si l'erreur ne se produit qu'une fois, sinon on appuiera sur 'A'.

Une erreur peut aussi être signalée lors de la lecture ou l'écriture sur une disquette;

```
Bad FAT
```

Ce message signifie que les données dans la 'File Allocation Table' renvoient à un secteur non-existant. Il se peut que la disquette ait été formatée incorrectement ou pas du tout. Si cette erreur se produit, l'information sur la disquette n'est pas récupérable et donc inutile. Pour pouvoir réutiliser cette disquette, le seul moyen est de la formater à nouveau.

MSX-DOS peut aussi afficher d'autre messages:

```
Bad command or file name
```

La commande spécifiée n'existe pas ou le mot spécifié ne renvoie pas à un fichier existant.

```
Drive name (A,B)?
```

Après la commande FORMAT le système demande quelle disquette doit être formatée.

```
Format complete
```

Dès que le formatage est achevé, ce message apparaît à l'écran.

```
Invalid date
Enter new date:
```

Ce message apparaît lorsqu'une date est incorrecte au démarrage ou après la commande DATE.

```
.. copied
```

Dès qu'un fichier a été copié, l'ordinateur le signale.

```
Invalid time
Enter new time:
```

Ce message apparaît lorsqu'on commet une erreur à l'entrée de l'heure. Pas sur tous les ordinateurs MSX.

```
Strike a key when ready
```

Ce message apparaît sur l'écran chaque fois qu'il faut changer de disquette.

```
Terminate batch job (Y/N)?
```

Ce message apparaît lorsqu'un fichier de commandes est interrompu par <CONTROL>-C. 'Y' arrêtera le fichier séquentiel, tandis que 'N' le poursuivra avec la commande suivante.

**Mots réservés**

MSX-DOS connaît un certain nombre de mots réservés. En principe les noms de fichiers .COM et .BAT sont des mots réservés qu'il vaut mieux ne pas utiliser pour des fichiers de données

ou de programmes sur la même disquette. Les commandes MSX-DOS ne peuvent pas être utilisées comme nom de fichier, ainsi que les mots suivants.

| | |
|---|---|
| AUX | MODE |
| BASIC | NUL |
| CON | PAUSE |
| COPY | PRN |
| DATE | REM |
| DEL | REN |
| DIR | RENAME |
| ERASE | TIME |
| FORMAT | TYPE |
| LST | VERIFY |

Un certain nombre de noms de fichier complets ne peuvent pas non plus être utilisés:

AUTOEXEC.BAS
AUTOEXEC.BAT
COMMAND.COM
MSXDOS.SYS
DOSHLP.COM

Il faut de même éviter que les fichiers de données que vous avez créés, par exemple des fichiers d'adresses, soient munis des extensions .COM ou .BAT. Ces extensions signifient pour MSX-DOS à la fois programme en langage machine et fichier de commandes.

En outre, il est recommandé de ne pas utiliser en MSX-DOS les mots réservés de MSX BASIC, ceci pour éviter plus tard une certaine confusion.

**Autres procédés de départ**

Quand l'unité de disquettes, l'écran (téléviseur ou moniteur) et l'ordinateur MSX sont mis en marche, le procédé de départ normal de MSX est le suivant:

1. Il est d'abord vérifié s'il y a une disquette dans l'unité A. Si ce n'est pas le cas, MSX-Disk BASIC est activé.

2. S'il se trouve bien une disquette dans l'unité A, le fichier MSXDOS.SYS est cherché. S'il ne se trouve pas sur la disquette, MSX-Disk BASIC est activé et le fichier AUTOEXEC.BAS, s'il se trouve sur la disquette, est exécuté comme un programme BASIC.

3. Si MSXSYS.DOS se trouve sur la disquette, le fichier COMMAND.COM est lu. Si ce fichier n'est pas présent, la de-

mande d'insérer une disquette avec le fichier COMMAND.COM est affichée.

4. Ensuite le fichier AUTOEXEC.BAT est exécuté comme un fichier de commandes (voir chapitre 7). S'il ne se trouve pas sur la disquette, la date est demandée (voir chapitre 5).

Le procédé décrit ci-dessus vaut quand l'ordinateur MSX est allumé et que vous appuyez sur le bouton RESET.

**Procédé de démarrage avec SHIFT**

Si vous allumez l'unité ou les unités de disques, l'écran (téléviseur ou moniteur) et l'ordinateur MSX, et puis appuyez sur la touche SHIFT, l'interface de l'unité de disques ne sera pas reconnue et MSX-BASIC, au lieu de MSX-Disk BASIC, sera activé.

C'est utile quand vous voulez travailler en MSX-BASIC au lieu de MSX-Disk BASIC, puisque vous n'avez pas à enlever l'interface de l'unité de disquettes de l'ordinateur MSX.

Avec les ordinateurs MSX ayant une unité de disques incorporée, vous êtes obligé de démarrer de cette façon si vous voulez travailler en MSX BASIC au lieu de MSX-Disk BASIC, puisque dans ce cas il est impossible d'enlever l'interface.

Ce procédé s'applique également si l'ordinateur MSX est allumé et que vous appuyez sur la touche RESET et simultanément sur la touche SHIFT.

**Procédé de démarrage avec CTRL**

Si vous allumez l'unité ou les unités de disques, l'écran (téléviseur ou moniteur) et l'ordinateur MSX, et puis appuyez sur la touche CTRL, seule l'unité A est reconnue.

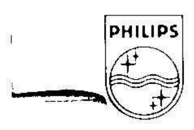
L'indication de l'unité B dans les commandes MSX-DOS ou dans les instructions MSX-Disk BASIC amène à un signal d'erreur. L'avantage de ce procédé est que la capacité de mémoire disponible en MSX-Disk BASIC est plus grande. Cela peut être très utile dans les programmes demandant une plus grande capacité de mémoire que la capacité disponible en MSX-Disk BASIC après le procédé de démarrage normal.

Le procédé décrit ci-dessus vaut également si l'ordinateur MSX est allumé et que vous appuyez sur le bouton RESET et simultanement sur la touche CTRL.

# INDEX

Que faire avec un micro ordinateur chez soi?
Comment tirer parti des extraordinaires possibilités du BASIC MSX?
Ce livre d'accès facile et pratique sera pour vous un excellent outil d'apprentissage de l'informatique. Spécialement conçu pour les néophytes, il vous permettra de vous initier à la programmation en BASIC, d'utiliser au mieux vos périphériques.

Avec le système d'exploitation MSX-DOS vous avez franchi un pas important vers une configuration informatique plus professionnel. Ce système d'exploitation vous permet d'écrire des programmes en Assembler, en C ou en d'autres langages de programmation. De plus, MSX-DOS vous donne la possibilité de passer sur votre ordinateur MSX un grand nombre de progiciels professionnels basés sur CP/M.
De cette façon, MSX-DOS étend considérablement la gamme d'application de votre ordinateur MSX et le convertit en une machine appropriée à beaucoup de tâches professionnelles.







8811 8622 518 81 211